Optimal Verification of Operations on Dynamic Sets

Charalampos Papamanthou
Brown University
Providence RI

Roberto Tamassia
Brown University
Providence RI

Nikos Triandopoulos*
RSA Laboratories
Cambridge MA

Abstract

We study the verification of *set operations* in the model of *authenticated data structures*, namely the problem of cryptographically checking the correctness of outsourced set operations performed by an untrusted *server* over a dynamic collection of sets that are owned (and updated) by a trusted *source*. We present a new authenticated data structure scheme that allows any entity to *publicly* verify the correctness of primitive sets operations such as *intersection*, *union*, *subset* and *set difference*. Based on a novel extension of the security properties of *bilinear-map accumulators* as well as on a primitive called *accumulation tree*, our authenticated data structure is the first to achieve *optimal* verification and proof complexity (i.e., only proportional to the size of the query parameters and the answer), as well as *optimal* update complexity (i.e., constant), and without bearing any extra asymptotic space overhead. Queries (i.e., constructing the proof) are also efficient, adding a *logarithmic* overhead to the complexity needed to compute the actual answer. In contrast, existing schemes entail high communication and verification costs or high storage costs as they recompute the query over authentic data or precompute answers to all possible queries. Applications of interest include efficient verification of keyword search and database queries. We base the security of our constructions on the *bilinear q-strong Diffie-Hellman* assumption.

Keywords: Authenticated structures, outsourced verifiable computation, pairing-based cryptography.

1 Introduction

Providing integrity guarantees in third-party data management settings is an active area of research, especially in view of the growth in usage of cloud computing (e.g., Amazon web services). In such settings, verifying the correctness of outsourced computations performed over remotely stored data becomes a crucial property for the trustworthiness of cloud services. Such a verification process should incur minimal overheads to the clients or otherwise the benefits of computation outsourcing are dismissed; ideally, computations should be verified without having to locally rerun them or to utilize too much extra cloud storage.

In this paper, we study the verification of outsourced operations on general *sets* and consider the following problem. Assuming that a dynamic collection of m sets $S_1, S_2, \ldots, S_m$ is remotely stored at an untrusted server, we wish to *publicly* verify primitive queries on these sets, such as *intersection*, *union* and *set difference*. For example, for the query requesting the intersection of t sets specified by indices $i_1, i_2, \ldots, i_t$ between 1 and m, we wish to design techniques that allow any client to cryptographically check the correctness of the returned intersection $S_{i_1} \cap S_{i_2} \cap \ldots \cap S_{i_t}$. In addition, we wish the verification of any set operation be *operation-sensitive*, meaning that the required complexity depends only on the (description and outcome of the) operation, and not on the sizes of the involved sets. For example, if $|S_{i_1} \cap S_{i_2} \cap \ldots \cap S_{i_t}| = \delta$ then we would like the verification cost to be proportional to $t + \delta$. This achieves optimality, as the query and the answer require $O(t + \delta)$ complexity. Applications of interest include computations related to *keyword search* and *database queries*, which boil down to set operations.

*Research performed while the author was at Boston University.

Relation to outsourced verifiable computation. Recent works on *outsourced verifiable computation* [2, 11, 17] achieve operation-sensitive verification of general functionalities. Although such approaches completely cover set operations as a special case, clearly meeting our goal with respect to optimal verifiability, they are inherently inadequate to meet our other goals with respect to *public verifiability* and *dynamic updates*, both important properties in the context of data querying. Indeed, the works in [2, 11, 17] are primarily designed to provide secrecy of the outsourced computations, and as such, the client makes use of some secret information to outsource the computation as a circuit and in an encrypted form. This secret information is also used in the verifying computation, therefore effectively supporting only one verifier; instead, we seek for schemes that allow any client to query the sets collection and verify the returned results. Finally, in [2, 11, 17], the description of the circuit is fixed at the initialization of the scheme, therefore effectively supporting no updates in the outsourced data; instead, we seek for schemes supporting efficient updates.

We accordingly study our problem in the model of *authenticated data structures*. A typical setting where an authenticated data structure can be employed involves three participating entities, usually referred to as *three-party* model [39] (see Corollary 1): A trusted party called *source*, owns a data structure (a collection of sets in our case), that is replicated along with some cryptographic information to one or more untrusted parties, called *servers*. *Clients* issue data structure queries to the servers and wish to *publicly* verify the answers received by the servers, based only on the trust they have in the source. This trust is conveyed through a time-stamped signature on a *digest* of the data structure (e.g., the roothash of a Merkle tree), that is made available by the source. Updates are performed both by the source and the server. Variations of this model include (a) the *two-party* model [33], where the source keeps only a small state (i.e., the digest) and performs both the updates and the queries/verifications—this model is *directly comparable* to the model of outsourced verifiable computation [2, 11, 17], see Corollary 2; (b) the *memory checking* model [7], where a memory of n cells being accessed through read/write operations is to be verified. However, the absence of the notion of *proof computation* in memory checking (it is just a storage device) as well as the requirement for *public verifiability*[1] in authenticated data structures make these models fundamentally different.

Achieving operation-sensitive verification. In this work, we design authenticated data structures for the verification of set operations in an operation-sensitive manner, that is, with proof and verification complexity depending only on the description and outcome of the operation and not on the size of the sets involved. Conceptually, this property is similar to the property of *super-efficient verification* that has been studied in certifying algorithms [22] and certification data structures [20, 40] (as well as in the context of *outsourced verifiable computation* [2, 11, 17]), where an answer can be verified in complexity asymptotically less than the complexity required to produce it. Whether the above optimality property is achievable for set operations (with linear storage) was posed as an open problem in [13]. We close this problem in the affirmative.

All existing schemes for verifying outsourced set operations fall into the following two rather straightforward and highly inefficient solutions. Either short proofs for the answer of every possible set operation query are precomputed allowing for highly imbalanced schemes: optimal verification overhead at the client comes at the cost of exponential storage and update overheads at the source and the server—an undesirable trade-off, as it is against storage outsourcing. Or integrity proofs for all the elements of the sets participating in the query are given to the client who locally verifies the set operation: in this case verification complexity can be linear in the problem size—an undesirable feature, as it is against computation outsourcing.

Intuition of our construction. We achieve optimal verification complexity by departing from the above approaches as follows. We first reduce the problem of verifying set operations to the problem of *verifying the validity of some more primitive relations on sets*, namely *subset containment* and *set disjointness*. Then for each such primitive relation we employ a corresponding cryptographic primitive to optimally verify its validity. In particular, we extend the bilinear-map accumulator to optimally verify *subset containment*, inspired by [35]. We then employ the extended Euclidean algorithm over polynomials in combination with subset

[1] Memory checking might require *secret* memory, e.g., see the PRF construction in [7].

Table 1: Asymptotic *access* and *group* complexities of various schemes defined by algorithms $\{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$, for a sets collection data structure of m sets: The sum of sizes of all the sets is M, $0 < \epsilon < 1$ is a constant, "Generic CR" stands for "Generic Collision Resistance" and upd is the update information, output by $\mathsf{update}()$. We show complexities for an intersection query on $t = O(1)$ sets, outputting an intersection δ elements. All sizes of the intersected and updated sets are $\Theta(n)$.

	$\mathsf{setup}()$	$\mathsf{update}()$ & $\mathsf{refresh}()$	$\mathsf{query}()$	$\mathsf{verify}()$ & proof $\Pi(q)$	upd	**assumption**
[13, 41]	$m + M$	$\log n + \log m$	$n + \log m$	$n + \log m$	1	Generic CR
[29]	$m + M$	$m + M$	n	n	n	Strong RSA
[32]	$m^t + M$	m^t	1	δ	m^t	D. Log
this	$m + M$	1	$n \log^3 n + m^\epsilon \log m$	δ	1	Bilinear q-Strong DH

containment proofs to provide a novel optimal verification test for *set disjointness*. The intuition behind our technique is that disjoint sets can be represented by polynomials mutually indivisible, therefore there exist other polynomials so that the sum of their pairwise products equals to one—this is the test to be used in the proof. However, transmitting (and processing) these polynomials is bandwidth (and time)-prohibitive and does not lead to operation-sensitivity. Taking advantage of bilinearity properties, we can compress their coefficients in the exponent and still use them in a meaningful way, i.e., compute an internal product. This is why although using a conceptually simpler RSA accumulator [6] would lead to a mathematically sound solution, a bilinear-map accumulator [31] is essential for achieving the desired complexity goal.

Authenticated data structure scheme. To formally describe our solutions and prove their properties we use an *authenticated data structure scheme*, which (informally) comprises a collection of algorithms $\{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$ such that (a) $\mathsf{genkey}()$ produces the secret and public key of the system; (b) $\mathsf{setup}()$ initializes (i.e., computes) the authenticated data structure $\mathsf{auth}(D)$, on input a plain data structure D; (c) *having* access to the secret key, $\mathsf{update}()$ updates the authenticated data structure digest (e.g., the roothash of a Merkle tree), so that it could be used later for *query verification*; (d) *without having* access to the secret key, $\mathsf{refresh}()$ updates the authenticated data structure as a whole, so that it could be used later for *query execution*; (e) $\mathsf{query}()$ computes cryptographic proofs $\Pi(q)$ for answers $\alpha(q)$ to data structure queries q; (f) $\mathsf{verify}()$ processes the proof $\Pi(q)$ and the answer $\alpha(q)$ and either *accepts* or *rejects* the answer. Note that both $\mathsf{query}()$ and $\mathsf{verify}()$ are required to have no access to the secret key. The formal definition of all the algorithms above and the properties they should satisfy (correctness/security) are given in Definition 1. We note that both a three-party protocol [39] (see Corollary 1) and a two-party protocol [33] (see Corollary 2) can be realized via an authenticated data structure scheme.

Complexity model. To explicitly measure complexity with respect to the number of primitive cryptographic operations, without considering the dependency on the security parameter, we adopt the complexity model used in memory checking [7, 14].[2] The *access* complexity of an algorithm is defined as the *number of memory accesses* this algorithm performs on the authenticated data structure stored in an indexed memory of n cells, in order for the algorithm to complete its execution. We require that each memory cell can store up to $O(\mathsf{poly}(\log n))$ bits, a word size used in Blum's original memory checking work [7] but also in subsequent work [14]. For example, a Merkle tree [27] has $O(\log n)$ update access complexity since the update algorithm needs to read and write $O(\log n)$ memory cells of the authenticated data structure, each cell storing exactly one hash value. In our context, the *group* complexity of a data collection (e.g., proof group complexity) is defined as the number of elementary data objects (e.g., group elements or elements in $\mathbb{Z}_p$) contained in that object. Whenever it is clear from the context, we omit the terms "access" and "group".

Related work. The great majority of authenticated data structures involve the use of cryptographic hash-

[2] This model was only implicitly used in the literature of the authenticated data structures: here, "access complexity" is used instead of "query complexity" (it also avoids ambiguity when referring to the $\mathsf{query}()$ algorithm of the authenticated data structure).

ing [3, 7, 19, 21, 25, 26, 30] or other primitives [18, 34, 35] to hierarchically compute over the outsourced data one or more secure digests. Most of these schemes incur verification costs that are proportional to the time spent to produce the query answer, thus not achieving operation sensitivity. Some bandwidth-optimal and operation-sensitive solutions for verification of various (e.g., range search) queries appear in [3, 20].

Despite the fact that privacy-related problems for set operations have been extensively studied in the cryptographic literature (e.g., [9, 15]), existing work on the integrity dimension of set operations appears mostly in the database literature. In [13], the importance of coming up with an operation-sensitive scheme is identified. In [29], possibly the closest in context work to ours, set intersection, union and difference are authenticated with linear verification and proof costs. Same linear asymptotic bounds are achieved in [41]. In [32], a different approach is taken: In order to achieve operation-sensitivity, expensive pre-processing and exponential space are required (i.e., answers to all possible queries are signed). Finally, related to our work are non-membership proofs, both for the RSA [23] and the bilinear-map [4, 12] accumulators.

Contributions. Our main result (Theorem 1) is the first authenticated data structure that achieves *optimal* verification of the sets operations *intersection*, *union*, *subset* and *set difference* (as well as optimal updates). This closes an open problem posed in [13]. Our scheme is proved secure under the bilinear extension of the q-strong Diffie-Hellman assumption (see, e.g., [8]). In the Appendix, we give applications (Section 11) of our construction to the authentication of keyword-search queries and database queries (e.g., equi-join), as long as some experimental results showing practicality (Section 12). A comparison of our work with existing schemes appears in Table 1. The detailed proofs of our results appear in the Appendix.

2 Preliminaries

In the following, we denote with k the security parameter and with $\mathsf{neg}(k)$ a negligible function[3].

Bilinear pairings. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p, generated by g. Let also $\mathcal{G}$ be a cyclic multiplicative group with the same order p and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathcal{G}$ be a bilinear pairing with the following properties: (1) Bilinearity: $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$; (2) Non-degeneracy: $e(g, g) \neq 1$; (3) Computability: There is an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in \mathbb{G}$. We denote with $(p, \mathbb{G}, \mathcal{G}, e, g)$ the bilinear pairings parameters, output by a PPT algorithm on input 1^k.

The bilinear-map accumulator. Let $(p, \mathbb{G}, \mathcal{G}, e, g)$ be a tuple of bilinear pairing parameters. The bilinear-map accumulator [31] is an efficient way to provide short proofs of (non-)membership for elements that (do not) belong to a set: It accumulates elements in $\mathbb{Z}_p^*$ and the accumulated value is an element in $\mathbb{G}$. For a set of elements $\mathcal{X}$ in $\mathbb{Z}_p^*$ the accumulation value $\mathsf{acc}(\mathcal{X})$ is defined as $\mathsf{acc}(\mathcal{X}) = g^{\prod_{x \in \mathcal{X}}(x+s)}$, where $s \in \mathbb{Z}_p^*$ is a randomly chosen value that constitutes the trapdoor in the scheme. We note here that $\mathsf{acc}(\mathcal{X})$ can be constructed by only using $\mathcal{X}$ and $g, g^s, g^{s^2}, \ldots, g^{s^q}$ (polynomial interpolation). The proof for *subset containment* of a set $\mathcal{S} \subseteq \mathcal{X}$—for $|\mathcal{S}| = 1$, this is a proof of membership—is the witness $(\mathsf{W}_{\mathcal{S},\mathcal{X}}, \mathcal{S})$ where

$$\mathsf{W}_{\mathcal{S},\mathcal{X}} = g^{\prod_{x \in \mathcal{X}-\mathcal{S}}(x+s)} . \tag{1}$$

A verifier can test subset containment for $\mathcal{S}$ by checking the relation $e(\mathsf{W}_{\mathcal{S},\mathcal{X}}, g^{\prod_{x \in \mathcal{S}}(x+s)}) \stackrel{?}{=} e\left(\mathsf{acc}(\mathcal{X}), g\right)$. Proving the security of the accumulator properties that we are using here requires the *bilinear* q-strong Diffie-Hellman assumption, a *slightly* stronger assumption than the q-strong Diffie-Hellman assumption [8].[4]

Assumption 1 (Bilinear q-strong Diffie-Hellman assumption) *Let k be the security parameter and let $(p, \mathbb{G}, \mathcal{G}, e, g)$ be a uniformly randomly generated tuple of bilinear pairings parameters. Given the elements $g, g^s, \ldots, g^{s^q} \in \mathbb{G}$ for some s chosen at random from $\mathbb{Z}_p^*$, where $q = \mathsf{poly}(k)$, there is no polynomial-time algorithm that can output the pair $(a, e(g, g)^{1/(s+a)}) \in \mathbb{Z}_p^* \times \mathcal{G}$ except with negligible probability $\mathsf{neg}(k)$.*

[3] Function $f : \mathbb{N} \rightarrow \mathbb{R}$ is $\mathsf{neg}(k)$ iff for any nonzero polynomial $p(k)$ there exits N such that for all $k > N$ it is $f(k) < 1/p(k)$.

[4] However, proving just *collision resistance* of the accumulator requires the plain q-strong Diffie-Hellman assumption [31].

We continue with proving security of subset witnesses—the subset witnesses also appeared simultaneously (without a proof though) in [10]. We note that this proof is a generalization of the one in [31]:

Lemma 1 (Proving subsets) *Let k be the security parameter and let $(p, \mathbb{G}, \mathcal{G}, e, g)$ be a uniformly randomly generated tuple of bilinear pairings parameters. Given the elements $g, g^s, \ldots, g^{s^q} \in \mathbb{G}$ for some s chosen at random from $\mathbb{Z}_p^*$ and a set of elements $\mathcal{X}$ in $\mathbb{Z}_p$ ($q \geq |\mathcal{X}|$), suppose there is a polynomial-time algorithm that finds $\mathcal{S}$ and W such that $\mathcal{S} \nsubseteq \mathcal{X}$ and $e(\mathsf{W}, g^{\prod_{x \in \mathcal{S}}(x+s)}) = e(\mathsf{acc}(\mathcal{X}), g)$. Then there is a polynomial-time algorithm for breaking the bilinear q-strong Diffie-Hellman assumption.*

We now define an *authenticated data structure scheme*, by describing the following six algorithms:

Definition 1 (Authenticated data structure scheme) *Let D be* any *data structure supporting queries and updates. We denote with $\mathsf{auth}(D)$ the authenticated data structure and with d the digest of the authenticated data structure, i.e., a constant-size description of D. An* authenticated data structure scheme *$\mathcal{A}$ is a collection of the following six polynomial-time algorithms $\{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$: **(1)** $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^k)$: Outputs secret and public keys sk and pk, given the security parameter k; **(2)** $\{\mathsf{auth}(D_0), d_0\} \leftarrow \mathsf{setup}(D_0, \mathsf{sk}, \mathsf{pk})$: Computes the authenticated data structure $\mathsf{auth}(D_0)$ and the respective digest of it, d_0, given a plain data structure D_0, the secret key sk and the public key pk; **(3)** $\{D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}, \mathsf{upd}\} \leftarrow \mathsf{update}(u, D_h, \mathsf{auth}(D_h), d_h, \mathsf{sk}, \mathsf{pk})$: On input an update u on data structure D_h, the authenticated data structure $\mathsf{auth}(D_h)$ and the digest d_h, it outputs the updated data structure D_{h+1} along with $\mathsf{auth}(D_{h+1})$, the updated digest d_{h+1} and some relative information upd. It requires the secret key for execution; **(4)** $\{D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}\} \leftarrow \mathsf{refresh}(u, D_h, \mathsf{auth}(D_h), d_h, \mathsf{upd}, \mathsf{pk})$: On input an update u on data structure D_h, the authenticated data structure $\mathsf{auth}(D_h)$, the digest d_h and relative information upd output by $\mathsf{update}()$, it outputs the updated data structure D_{h+1} along with $\mathsf{auth}(D_{h+1})$ and the updated digest d_{h+1},* without *having the secret key as input; **(5)** $\{\Pi(q), \alpha(q)\} \leftarrow \mathsf{query}(q, D_h, \mathsf{auth}(D_h), \mathsf{pk})$: On input a query q on data structure D_h and $\mathsf{auth}(D_h)$ this algorithm returns the answer to the query $\alpha(q)$, along with a proof $\Pi(q)$; **(6)** $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \alpha(q), \Pi(q), d_h, \mathsf{pk})$: On input a query q, an answer $\alpha(q)$, a proof $\Pi(q)$, a digest d_h and pk, it outputs either "accept" or "reject".*

Let now $\{\mathsf{accept}, \mathsf{reject}\} = \mathsf{check}(q, \alpha(q), D_h)$ be a method that decides whether $\alpha(q)$ is a correct answer for query q on data structure D_h. There are two properties that an authenticated data structure scheme should satisfy, i.e., *correctness* and *security* (intuition follows from signature schemes definitions):

Definition 2 (Correctness of authenticated data structure scheme) *Let $\mathcal{A}$ be an authenticated data structure scheme $\{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$. We say that the authenticated data structure scheme $\mathcal{A}$ is correct if, for all $k \in \mathbb{N}$, for all $\{\mathsf{sk}, \mathsf{pk}\}$ output by algorithm $\mathsf{genkey}()$, for all $D_h, \mathsf{auth}(D_h), d_h$ output by one invocation of $\mathsf{setup}()$ followed by polynomially-many invocations of $\mathsf{refresh}()$, where $h \geq 0$, for all queries q and for all $\Pi(q), \alpha(q)$ output by $\mathsf{query}(q, D_h, \mathsf{auth}(D_h), \mathsf{pk})$, with all but negligible probability, whenever $\mathsf{check}(q, \alpha(q), D_h)$ accepts, so does $\mathsf{verify}(q, \Pi(q), \alpha(q), d_h, \mathsf{pk})$.*

Definition 3 (Security of authenticated data structure scheme) *Let $\mathcal{A}$ be an authenticated data structure scheme $\{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$, k be the security parameter, $\nu(k)$ be a negligible function and $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^k)$. Let also Adv be a polynomially-bounded adversary that is only given pk. The adversary has unlimited access to all algorithms of $\mathcal{A}$, except for algorithms $\mathsf{setup}()$ and $\mathsf{update}()$ to which he has only oracle access. The adversary picks an initial state of the data structure D_0 and computes $D_0, \mathsf{auth}(D_0), d_0$ through oracle access to algorithm $\mathsf{setup}()$. Then, for $i = 0, \ldots, h = \mathsf{poly}(k)$, Adv issues an update u_i in the data structure D_i and outputs D_{i+1}, $\mathsf{auth}(D_{i+1})$ and d_{i+1} through oracle access to algorithm $\mathsf{update}()$. Finally the adversary picks an index $0 \leq t \leq h+1$, a query Q, an answer $\alpha(Q)$ and a proof $\Pi(Q)$. We say that the authenticated data structure scheme $\mathcal{A}$ is secure if for all $k \in \mathbb{N}$, for all $\{\mathsf{sk}, \mathsf{pk}\}$ output by algorithm $\mathsf{genkey}()$, and for all polynomially-bounded adversaries Adv it is*

$$\Pr\left[\begin{array}{ll} \{Q, \Pi(Q), \alpha(Q), t\} \leftarrow \mathsf{Adv}(1^k, \mathsf{pk}); & \mathsf{accept} \leftarrow \mathsf{verify}(Q, \alpha(Q), \Pi(Q), d_t, \mathsf{pk}); \\ & \mathsf{reject} = \mathsf{check}(Q, \alpha(Q), D_t). \end{array}\right] \leq \nu(k)\,. \quad (2)$$

3 Construction and algorithms

The data structure for which we design an authenticated data structure scheme for is called *sets collection* and is a generalization of the *inverted index* [5]. We describe it in detail in the following paragraph.

Sets collection. The *sets collection* data structure consists of m sets, denoted with $S_1, S_2, \ldots, S_m$, each containing elements from a universe $\mathcal{U}$. Without loss of generality we assume that our universe $\mathcal{U}$ is the set of nonnegative integers in the interval $[m+1, p-1]$, where p is k-bit prime, m is the number of the sets in our collection that has bit size $O(\log k)$ and k is the security parameter.[5] Every set S_i does not contain duplicate elements, however an element x can appear in more than one set. Each set is sorted. The space usage of the sets collection is $O(m+M)$, where M is the sum of the sizes of the sets.

In order to get some intuition, we can view the sets collection as an *inverted index*. The elements are pointers to documents and each set S_i corresponds to a term w_i in the dictionary, containing the pointers to documents where term w_i appears. In this example, m is the number of terms being indexed, which is typically in the hundreds of thousands, while M is the number of documents being indexed, which is in the billions. For the sake of generality, we refer to *elements*, instead of documents, and to *sets*, instead of terms.

The operations supported by the sets collection data structure consist of *updates* and *queries*. An update is either an *insertion* of an element into a set or a *deletion* of an element from a set. An update on a set of size n takes $O(\log n)$ time. For simplicity, we assume that the number m of sets does not change. A query is one of the following standard set operations: (1) *Intersection:* Given indices $i_1, i_2, \ldots, i_t$, return set $\mathsf{I} = S_{i_1} \cap S_{i_2} \cap \ldots \cap S_{i_t}$; (2) *Union:* Given indices $i_1, i_2, \ldots, i_t$, return set $\mathsf{U} = S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_t}$; (3) *Subset query:* Given indices i and j, return true if $S_i \subseteq S_j$ and false otherwise; (4) *Set difference:* Given indices i and j, return the set $\mathsf{D} = S_i - S_j$. Finally, and for the remainder of the paper, we denote with δ the size of the answer to a query operation, i.e., δ is equal to the size of I, U, or D. For a subset query, δ is $O(1)$.

In order to verify the integrity of the answers to set operation (under set updates), we are going to describe an authenticated data structure scheme $\mathcal{ASC}$ (see Definition 1) for the *sets collection* data structure. Our goal is to have no additional asymptotic overhead in the communication and verification complexity. I.e., for a query with t parameters and answer size δ, we want the proof and its verification to have optimal complexity $O(t+\delta)$. Also, we want algorithms $\mathsf{query}()$, $\mathsf{update}()$ and $\mathsf{refresh}()$ to have low complexity.

Algorithms of the scheme. We now describe the algorithms of an authenticated data structure scheme $\mathcal{ASC}$ (Definition 1) for a sets collection data structure and prove they satisfy the complexities of Table 1.

Algorithm $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^k)$**:** Pick bilinear pairings parameters $(p, \mathbb{G}, \mathcal{G}, e, g)$ and an element $s \in \mathbb{Z}_p^*$ at random. Subsequently, an one-to-one function $h(\cdot) : \mathbb{G} \rightarrow \mathbb{Z}_p^*$ is used. This function just outputs the bit description of the elements of $\mathbb{G}$. Finally the algorithm outputs $\mathsf{sk} = s$ and $\mathsf{pk} = \{h(\cdot), (p, \mathbb{G}, \mathcal{G}, e, g), \{g^{s^i} : i = 1, \ldots, q\}\}$, where $q \geq \max\{m, \max_{i=1,\ldots,m}\{|S_i|\}\}$. The algorithm has $O(1)$ access complexity.

Algorithm $\{D_0, \mathsf{auth}(D_0), d_0\} \leftarrow \mathsf{setup}(D_0, \mathsf{sk}, \mathsf{pk})$**:** Let D_0 be our initial data structure, i.e., the sets $S_1, S_2, \ldots, S_m$. The authenticated data structure $\mathsf{auth}(D_0)$ is built as follows: First of all, for each set S_i, the accumulation value of S_i, $\mathsf{acc}(S_i) = g^{\prod_{x \in S_i}(s+x)}$, is computed (see Section 2).[6] Subsequently, the algorithm picks a constant $0 < \epsilon < 1$. Let T be a tree structure that has $l = \lceil 1/\epsilon \rceil$ levels and m leaves, numbered $1, 2, \ldots, m$, where m is the *number* of the sets of our sets collection data structure. Since this is a *constant-height* tree, the degree of any internal node of T is $O(m^\epsilon)$. We call such a tree an *accumulation tree*, which was originally introduced (combined with different cryptography) in [35]. For each node of the tree v, the algorithm recursively computes the *digest* $d(v)$ of v as follows: If v is a leaf corresponding to set S_i, where $1 \leq i \leq m$, the algorithm sets $d(v) = \mathsf{acc}(S_i)^{(s+i)}$ (accumulating $s+i$ in the exponent, under the constraint that $i \leq m$, is used to prove that S_i refers to $\mathsf{acc}(S_i)$). If node v is not a leaf, then

$$d(v) = g^{\prod_{w \in \mathcal{N}(v)}(s+h(d(w)))}, \tag{3}$$

[5] We could easily set our universe to be $\mathbb{Z}_p^*$ by using CRHFs, but we choose not to do so in sake of a cleaner presentation.

[6] The polynomial $\prod_{x \in S_i}(s+x)$ is called *characteristic polynomial* of set S_i in the literature (e.g., see [28]).

where $\mathcal{N}(v)$ denotes the set of children of node v. The algorithm outputs all the sets S_i as the data structure D_0, all the accumulation values $\mathsf{acc}(S_i)$ for $1 \leq i \leq m$, the tree T and all the digests $d(v)$ for all $v \in T$ as $\mathsf{auth}(D_0)$. Finally, the algorithm sets $d_0 = d(r)$,[7] where r is the root of T (i.e., the *digest* of the authenticated data structure is defined as in a Merkle tree). The access complexity of the algorithm is $O(m+M)$ (postorder traversal of T and computation of $\mathsf{acc}(S_i)$), where $M = \sum_{i=1}^{m} |S_i|$ and the group complexity of $\mathsf{auth}(D_0)$ is also $O(m+M)$ since the algorithm stores one digest per node of T, T has $O(m)$ nodes and there are also M elements contained in the stored sets, as part of $\mathsf{auth}(D_0)$.

Algorithm $\{D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}, \mathsf{upd}\} \leftarrow \mathsf{update}(u, D_h, \mathsf{auth}(D_h), d_h, \mathsf{sk}, \mathsf{pk})$**:** Suppose the update is "insert element $x \in \mathcal{U}$ into set S_i". Let v_0 be the leaf node of T referring to set i. Let $v_0, v_1, \ldots, v_l$ be the path in $T(\epsilon)$ from node v_0 to the root of the tree, where $l = \lceil 1/\epsilon \rceil$. The algorithm initially sets $d'(v_0) = \mathsf{acc}(S_i)^{x+s}$, i.e., it updates the accumulation value that corresponds to the updated set. Note that in the case of deleting x from S_i, the algorithm sets $d'(v_0) = \mathsf{acc}(S_i)^{(x+s)^{-1}}$. Then the algorithm sets

$$d'(v_j) = d(v_j)^{(h(d'(v_{j-1}))+s)(h(d(v_{j-1}))+s)^{-1}} \text{ for } j = 1, \ldots, l\,, \tag{4}$$

where $d(v_{j-1})$ is the previous digest of v_j and $d'(v_{j-1})$ is the updated digest of v_{j-1}.[8] All these values are stored by the algorithm after they have been computed. The algorithm also outputs the new digests $d'(v_{j-1})$ $(i = 1, \ldots, l)$ as the information upd along the path from the updated set to the root of the tree. Information upd also includes x and $d'(v_l)$. Also it sets $d_{h+1} = d'(v_l)$, i.e., the updated digest is the updated digest of the root of T. Finally the new authenticated data structure $\mathsf{auth}(D_{h+1})$ is computed as follows. Let $\mathsf{auth}(D_h)$ be the previous authenticated data structure that is input the algorithm. Overwrite the values $d(v_{j-1})$ $(j = 1, \ldots, l)$ with the new values $d'(v_{j-1})$ $(j = 1, \ldots, l)$ and output the updated structure. All the operations performed are proportional to $1/\epsilon$, therefore the access complexity of the algorithm is $O(1)$.

Algorithm $\{D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}\} \leftarrow \mathsf{refresh}(u, D_h, \mathsf{auth}(D_h), d_h, \mathsf{upd}, \mathsf{pk})$**:** Suppose the update is "insert element $x \in \mathcal{U}$ into set S_i". Let v_0 be the node of $T(\epsilon)$ referring to set S_i. Let $v_0, v_1, \ldots, v_l$ be the path in T from node v_0 to the root of the tree. The algorithm, for $j = 0, \ldots, l$, sets $d(v_j) = d'(v_j)$, i.e., it updates the digests that correspond to the updated path by using the information upd. Finally it outputs the updated sets collection as D_{h+1}, the updated digests $d(v_j)$ (along with the ones that belong to the nodes that are not updated) as $\mathsf{auth}(D_{h+1})$ and $d'(v_l)$ (contained in upd) as d_{h+1}.[9] The algorithm has $O(1)$ access complexity since it performs a number of operations proportional to $1/\epsilon$.

3.1 Queries and verification

In this section, we show how compact proofs for the answers to set queries (e.g., intersection, union) can be constructed using the authenticated sets collection data structure presented earlier. The proofs have optimal size $O(t+\delta)$, where t is the size of the query parameters (e.g., $t = 2$ for an intersection of two sets) and δ is the answer size (e.g., $\delta = 1$ if the intersection consists of one element). Our solutions use polynomial arithmetic. The following result is derived by using an FFT algorithm (e.g., see Preparata and Sarwate [37]—note that there is no requirement for existence of an n-th root of unity in $\mathbb{Z}_p$ for the algorithm to work) that computes the DFT in a finite field (e.g., $\mathbb{Z}_p$) for arbitrary n and with $O(n \log n)$ field operations.

Lemma 2 (Polynomial interpolation with FFT [37]) *Let $\prod_{i=1}^{n}(s + x_i) = \sum_{i=0}^{n} a_i s^i$ be a degree-n polynomial. The coefficients $a_n, a_{n-1}, \ldots, a_0$ can be computed with $O(n \log n)$ complexity, given $x_1, x_2, \ldots, x_n$.*

Lemma 2 describes how we can compute the coefficients $a_n, a_{n-1}, \ldots, a_0$ of a polynomial, given its roots. In the following, we give a result, related to *certifying algorithms* [22], stating that if the vector coefficients $\mathbf{a} = [a_n, a_{n-1}, \ldots, a_0]$ is claimed to be correct, it can be certified, with high probability, with

[7] Digest $d(r)$ is a "secure" succinct description of the set collection data structure. Namely, the accumulation tree protects the integrity of $\mathsf{acc}(S_i)$ and the accumulation value $\mathsf{acc}(S_i)$ protects the integrity of the elements contained in set S_i.

[8] Note that this algorithm is efficient because it has access to the secret key s.

[9] Note that information upd is not *required* for $\mathsf{refresh}()$ to execute but is used for efficiency. Namely, algorithm $\mathsf{refresh}()$ could compute the updated values $d(v_j)$ by doing polynomial interpolation, which would have $O(m^{\epsilon} \log m)$ complexity (see Lemma 2).

complexity less than $O(n \log n)$, i.e., without an FFT computation from scratch and given the vector of roots $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. This can be achieved with the following algorithm (not part of the $\mathcal{ASC}$ scheme):

Algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{certify}(\mathbf{a}, \mathbf{x}, \mathsf{pk})$**:** Pick a random $\kappa \in \mathbb{Z}_p^*$. If $\sum_{i=0}^{n} a_i \kappa^i = \prod_{i=1}^{m} (\kappa + x_i)$, then the algorithm outputs accept, else it outputs reject.

Lemma 3 (Verification of polynomial coefficients) *Let* $\mathbf{a} = [a_n, a_{n-1}, \ldots, a_0]$ *and* $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. *Algorithm* $\mathsf{certify}(\mathbf{a}, \mathbf{x}, \mathsf{pk})$ *has* $O(n)$ *complexity. Also, if* $\mathsf{accept} \leftarrow \mathsf{certify}(\mathbf{a}, \mathbf{x}, \mathsf{pk})$*, then* $a_n, a_{n-1}, \ldots, a_0$ *are the coefficients of the polynomial* $\prod_{i=1}^{n} (s + x_i)$ *with probability* $\Omega(1 - \mathsf{neg}(k))$.

Proof of accumulation values. Essential for our methods to work is the construction and the verification of proofs for the accumulation values $\mathsf{acc}(S_i)$, stored at leaf i of the tree T, for $i = 1, \ldots, m$. Such proofs are a collection of $O(1)$ accumulator witnesses: For accumulation value $\mathsf{acc}(S_i)$, the proof Π_i is the ordered sequence $\pi_1, \pi_2, \ldots, \pi_l$, where π_j is a tuple of a digest $d(.)$ and a witness that authenticates every node of the path $v_0, v_1, \ldots, v_l$ from the accumulation value $\mathsf{acc}(S_i)$ (we recall that v_0 stores the accumulation value $\mathsf{acc}(S_i)$) in the root v_l of T. Thus, item π_j is defined as $\pi_j = (d(v_{j-1}), \mathsf{W}_{v_{j-1}(v_j)})$, where

$$\mathsf{W}_{v_{j-1}(v_j)} = g^{\prod_{w \in \mathcal{N}(v_j) - \{v_{j-1}\}} (s + h(d(w)))}, \tag{5}$$

is the witness for a subset of *one* element (i.e., the element $h(d(v_{j-1}))$), as defined in Section 2. For simplicity, we set $\beta_j = d(v_{j-1})$ (note that $d(v_0) = \mathsf{acc}(S_i)^{s+i}$) and $\gamma_j = \mathsf{W}_{v_{j-1}(v_j)}$. Note that each π_j has group complexity $O(1)$ and can be constructed with $O(m^\epsilon \log m)$ complexity, using polynomial interpolation (due to the degree bound of an internal node of T and Lemma 2). Also, since proof Π_i consists of $O(1)$ such π_j tuples, we conclude that the proof Π_i for an accumulation value $\mathsf{acc}(S_i)$ can be constructed with $O(m^\epsilon \log m)$ complexity and has $O(1)$ group complexity. In the following we formally give the algorithms $\mathsf{queryTree}()$ and $\mathsf{verifyTree}()$ for the construction and the verification of such proofs respectively. Similar methods have been described in [35].

Algorithm $\{\Pi_i, \alpha_i\} \leftarrow \mathsf{queryTree}(i, D_h, \mathsf{auth}(D_h), \mathsf{pk})$**:** Let $v_0, v_1, \ldots, v_l$ be the path of T from the node storing $\mathsf{acc}(S_i)$ to the root of T. The algorithm sets $\Pi_i = (\pi_1, \pi_2, \ldots, \pi_l)$, where $\pi_j = (d(v_{j-1}), \mathsf{W}_{v_{j-1}(v_j)})$ and $\mathsf{W}_{v_{j-1}(v_j)}$ is given in Relation 5, computed by Lemma 2. The algorithm sets $\alpha_i = \mathsf{acc}(S_i)$ as an answer.

Algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verifyTree}(i, \alpha_i, \Pi_i, d_h, \mathsf{pk})$**:** Let $\Pi_i = (\pi_1, \pi_2, \ldots, \pi_l)$ be the proof, where $\pi_j = (\beta_j, \gamma_j)$. The algorithm outputs "reject" if one of the following is true (note that the verification algorithm is using the bilinear map function $e(.,.)$): (a) $e(\beta_1, g) \neq e(\alpha_i, g^s g^i)$; (b) $e(\beta_j, g) \neq e\left(\gamma_{j-1}, g^s g^{h(\beta_{j-1})}\right)$ for some $2 \leq j \leq l$; (c) $e(d_h, g) \neq e\left(\gamma_l, g^s g^{h(\beta_l)}\right)$.

Lemma 4 *Algorithm* $\mathsf{queryTree}()$ *has* $O(m^\epsilon \log m)$ *access complexity, outputting a proof of* $O(1)$ *group complexity. Moreover algorithm* $\mathsf{verifyTree}()$ *has* $O(1)$ *access complexity.*

We now give the security property that holds for this construction. Lemma 5 is going to be used as a building block of the proof of security (Theorem 1) of the whole scheme (see proof in the Appendix):

Lemma 5 *For any adversarially chosen proof* Π_i $(1 \leq i \leq m)$*, if algorithm* $\mathsf{verifyTree}(i, \alpha_i, \Pi_i, d_h, \mathsf{pk})$ *accepts, then* $\alpha_i = \mathsf{acc}(S_i)$ *with probability* $\Omega(1 - \mathsf{neg}(k))$.

In the following we describe the algorithms for an *intersection* and a *union* query in detail. The details of the the *subset* and the *set difference* query are in the Appendix (Section 9). The parameters of an intersection or a union query are t indices, namely the indices $i_1, i_2, \ldots, i_t$, with $1 \leq t \leq m$. To simplify the notation, we assume without loss of generality that these indices are $1, 2, \ldots, t$. We denote with n_i the size of set S_i $(i = 1, 2, \ldots, t)$ and we define $N = \sum_{i=1}^{t} n_i$. I.e., N is the total size of the sets involved in the intersection or the union and δ is the size of the intersection or union. Note, that in all cases $\delta = O(N)$ and that performing the actual operations has $O(N)$ complexity, by using a generalized merge.

Intersection query. Let $\mathsf{I} = S_1 \cap S_2 \cap \ldots \cap S_t = \{y_1, y_2, \ldots, y_\delta\}$. We express the correctness of the answer I to the intersection query by means of the following two conditions:

$$\textbf{Subset condition: } \mathsf{I} \subseteq S_1 \wedge \mathsf{I} \subseteq S_2 \wedge \ldots \wedge \mathsf{I} \subseteq S_t\,; \tag{6}$$

$$\textbf{Completeness condition: } (S_1 - \mathsf{I}) \cap (S_2 - \mathsf{I}) \cap \ldots \cap (S_t - \mathsf{I}) = \emptyset\,. \tag{7}$$

Note the completeness condition in Equation 7 is necessary since I should contain *all* the common elements. Given an intersection I, and for every set S_j, we define polynomial $P_j(s) = \prod_{x \in S_j - \mathsf{I}}(x + s)$, of degree n_j. We can now state the following lemma: (see proof in the Appendix):

Lemma 6 *Set I is the intersection of sets $S_1, S_2, \ldots, S_t$ if and only if there exist polynomials $q_1(s), q_2(s), \ldots, q_t(s)$ such that $q_1(s)P_1(s) + q_2(s)P_2(s) + \ldots + q_t(s)P_t(s) = 1$. Moreover, computing polynomials $q_1(s), q_2(s), \ldots, q_t(s)$ has $O(N \log^2 N \log\log N)$ complexity.*

We can now use Lemmata 2 and 6 to construct efficient proofs for both conditions in Equations 6 and 7:

Proof of subset condition. For each set S_j, $1 \leq j \leq t$, the *subset witnesses* $\mathsf{W}_{\mathsf{I},j} = g^{P_j(s)} = g^{\prod_{x \in S_j - \mathsf{I}}(x+s)}$ are computed, as defined in Section 2: Witness $\mathsf{W}_{\mathsf{I},j}$ will serve as a proof that I is a subset of set S_j. By using Lemma 2, computing each such witness has complexity $O(n_j \log n_j)$. Thus, the total complexity for computing all the t subset witnesses is $O(N \log N)$, where $N = \sum_{i=1}^{t} n_i$.[10]

Proof of completeness condition. Suppose $q_1(s), q_2(s), \ldots, q_t(s)$ are polynomials computed in Lemma 6 that satisfy $q_1(s)P_1(s) + q_2(s)P_2(s) + \ldots + q_t(s)P_t(s) = 1$. For $j = 1, \ldots, t$, the *completeness witnesses* $\mathsf{F}_{\mathsf{I},j} = g^{q_j(s)}$ are computed. By Lemma 6, computing the completeness witnesses has $O(N \log^2 N \log\log N)$ complexity. We can now formally define algorithms $\mathsf{query}()$ and $\mathsf{verify}()$ of the authenticated data structure scheme $\mathcal{ASC}$ and for the *intersection* query:

Algorithm $\{\Pi(q), \alpha(q)\} \leftarrow \mathsf{query}(q, D_h, \mathsf{auth}(D_h), \mathsf{pk})$**:** (*intersection*)

The query q is the set of indices $\{1, 2, \ldots, t\}$, requiring the intersection of $S_1, S_2, \ldots, S_t$. Let $\alpha(q) = \{y_1, y_2, \ldots, y_\delta\}$ be the intersection I. The proof $\Pi(q)$ consists of the following parts: **(1)** *Coefficients* $b_\delta, b_{\delta-1}, \ldots, b_0$ of polynomial $(s+y_1)(s+y_2)\ldots(s+y_\delta)$ associated with the intersection $\mathsf{I} = \{y_1, y_2, \ldots, y_\delta\}$, computed with $O(\delta \log \delta)$ complexity (Lemma 2), and of total group complexity $O(\delta)$; **(2)** *Accumulation values* $\mathsf{acc}(S_j)$ associated with the sets S_j, along with their respective proofs Π_j, output by calling algorithm $\mathsf{queryTree}(j, D_h, \mathsf{auth}(D_h), \mathsf{pk})$, for $j = 1, \ldots, t$, the computation of which has $O(tm^\epsilon \log m)$ complexity (Lemma 4). Moreover the group complexity of all Π_j is $O(t)$ (Lemma 4); **(3)** *Subset witnesses* $\mathsf{W}_{\mathsf{I},j}$ (see above) for $j = 1, \ldots, t$, the computation of which has $O(N \log N)$ complexity (Lemma 2). Also the subset witnesses have total group complexity $O(t)$; **(4)** *Completeness witnesses* $\mathsf{F}_{\mathsf{I},j} = g^{q_j(s)}$ (see above) for $j = 1, \ldots, t$, the computation of which has $O(N \log^2 N \log\log N)$ complexity (Lemma 6). The completeness witnesses have group complexity $O(t)$.

Algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \alpha(q), \Pi(q), d_h, \mathsf{pk})$**:** (*intersection*)

The verification algorithm performs the following steps: **(1)** First it uses $\mathbf{b} = [b_\delta, b_{\delta-1}, \ldots, b_0]$ and the answer $\alpha(q) = \{y_1, y_2, \ldots, y_\delta\}$, and calls $\mathsf{certify}(\mathbf{b}, \alpha(q), \mathsf{pk})$ in order to certify the validity of $b_\delta, b_{\delta-1}, \ldots, b_0$. This step, by Lemma 3, has $O(\delta)$ complexity. If $\mathsf{certify}()$ rejects, the algorithm outputs reject;[11] **(2)** Subsequently, the algorithm uses the proofs Π_j to verify the integrity of $\mathsf{acc}(S_j)$, for $j = 1, \ldots, t$, by using algorithm $\mathsf{verifyTree}(j, \mathsf{acc}(S_j), \Pi_j, d_h, \mathsf{pk})$. This step has $O(t)$ complexity (Lemma 4). If the verification fails for at least one of $\mathsf{acc}(S_j)$, it outputs reject; **(3)** Next, the *subset condition* is checked:[12]

$$e\left(\prod_{i=0}^{\delta}\left(g^{s^i}\right)^{b_i}, \mathsf{W}_{\mathsf{I},j}\right) \stackrel{?}{=} e\left(\mathsf{acc}(S_j), g\right) \text{ for } j = 1, \ldots, t\,. \tag{8}$$

[10] This is because $\sum n_j \log n_j \leq \log N \sum n_j = N \log N$.

[11] Algorithm $\mathsf{certify}()$ is used to achieve optimal verification and avoid an $O(\delta \log \delta)$ FFT computation from scratch.

[12] Computing the quantity $\prod_{i=0}^{\delta} g^{b_i s^i} = g^{(s+y_1)(s+y_2)\ldots(s+y_\delta)}$ has $O(\delta)$ complexity, and is done once.

This step has $O(t+\delta)$ complexity. If any of the above checks fails, the algorithm outputs reject; **(4)** Finally, the *completeness condition* is checked:

$$\prod_{j=1}^{t} e\left(\mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},j}\right) \stackrel{?}{=} e(g,g)\,. \tag{9}$$

If this relation holds, the algorithm accepts $\alpha(q)$ as the *correct* intersection. This step has $O(t)$ complexity.

We note that for Relation 9, it indeed holds $\prod_{j=1}^{t} e\left(\mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},j}\right) = e(g,g)^{\sum_{j=1}^{t} q_j(s)P_j(s)} = e(g,g)$ when all the subset witnesses $\mathsf{W}_{\mathsf{I},j}$, all the completeness witnesses $\mathsf{F}_{\mathsf{I},j}$ and all the sets accumulation values $\mathsf{acc}(S_j)$ have been computed *honestly*, since $q_1(s)P_1(s)+q_2(s)P_2(s)+\ldots+q_t(s)P_t(s)=1$. This is a required step for proving that the authenticated data structure is *correct*, as defined in Definition 2 (proof in the Appendix). We continue with describing the remaining queries such as *union*, *subset* and *set difference*.

Union query. The answer to a union query is the set $\mathsf{U} = S_1 \cup S_2 \cup \ldots \cup S_t = \{y_1, y_2, \ldots, y_\delta\}$. We express the correctness of the answer U to the union query by means of the following two conditions:

$$\textbf{Membership condition: } \forall y_i \in \mathsf{U}\ \exists j \in \{1,2,\ldots,t\} : y_i \in S_j\,; \tag{10}$$

$$\textbf{Superset condition: } (\mathsf{U} \supseteq S_1) \wedge (\mathsf{U} \supseteq S_2) \wedge \ldots \wedge (\mathsf{U} \supseteq S_t)\,. \tag{11}$$

Note that the *superset condition* is needed to make sure that no element has been excluded from the returned answer U. We now formally describe algorithms $\mathsf{query}()$ and $\mathsf{verify}()$ for the *union* query.

Algorithm $\{\Pi(q), \alpha(q)\} \leftarrow \mathsf{query}(q, D_h, \mathsf{auth}(D_h), \mathsf{pk})$**:** (*union*)

The query q is the union of the sets $S_1, S_2, \ldots, S_t$. Let $\alpha(q) = \{y_1, y_2, \ldots, y_\delta\}$ be the union U. The proof $\Pi(q)$ consists of the following parts: **(1)** *Coefficients* $b_\delta, b_{\delta-1}, \ldots, b_0$ of polynomial $(s+y_1)(s+y_2)\ldots(s+y_\delta)$ associated with the union $\mathsf{I} = \{y_1, y_2, \ldots, y_\delta\}$; **(2)** *Accumulation values* $\mathsf{acc}(S_j)$ associated with the sets S_j, along with their respective proofs Π_j, output by calling algorithm $\mathsf{queryTree}(j, D_h, \mathsf{auth}(D_h), \mathsf{pk})$, for $j = 1, \ldots, t$; **(3)** For each y_i, $i = 1, \ldots, \delta$, a *membership witness* W_{y_i, S_k} is provided (see Relation 1). This proves that y_i belongs to *some* set S_k for some $k = 1, \ldots, t$. These δ witnesses have $O(\delta)$ group complexity and constructing them has $O(N \log N)$ complexity (Lemma 2); **(4)** Finally, for each $j = 1, \ldots, t$ a *subset witness* $\mathsf{W}_{S_j, \mathsf{U}}$ is provided—as defined in Relation 1. This proves that U is a superset of S_j. These witnesses have group complexity $O(t)$. Constructing them has $O(N \log N)$ complexity (Lemma 2).

Algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \alpha(q), \Pi(q), d_h, \mathsf{pk})$**:** (*union*)

The verification algorithm performs the following steps: **(1)** First it uses $\mathbf{b} = [b_\delta, b_{\delta-1}, \ldots, b_0]$ and the answer $\alpha(q) = \{y_1, y_2, \ldots, y_\delta\}$, and calls $\mathsf{certify}(\mathbf{b}, \alpha(q), \mathsf{pk})$ in order to certify the validity of $b_\delta, b_{\delta-1}, \ldots, b_0$; **(2)** Subsequently, the algorithm uses the proofs Π_j to verify the integrity of $\mathsf{acc}(S_j)$, for $j = 1, \ldots, t$, by using algorithm $\mathsf{verifyTree}(j, \mathsf{acc}(S_j), \Pi_j, d_h, \mathsf{pk})$. If the verification fails for at least one of $\mathsf{acc}(S_j)$, it outputs reject; **(3)** Then the algorithm verifies that each element y_i ($i = 1, \ldots, \delta$) of the reported union belongs to some set S_k, for some $k = 1, \ldots, t$ ($O(\delta)$ complexity). This is done by checking that the following relation holds $e(\mathsf{W}_{y_i, S_k}, g^{y_i} g^s) = e(\mathsf{acc}(S_k), g)$ for all $i = 1, \ldots, \delta$ (else output reject); **(4)** Finally, the algorithm verifies that all sets of the query are *subsets* of the union, by checking the following conditions:

$$e\left(\mathsf{W}_{S_j, \mathsf{U}}, \mathsf{acc}(S_j)\right) \stackrel{?}{=} e\left(\prod_{i=0}^{\delta} \left(g^{s^i}\right)^{b_i}, g\right) \text{ for } j = 1, \ldots, t\,.$$

If any of the above checks fails, reject is output. Else $\{y_1, y_2, \ldots, y_\delta\}$ is accepted as the correct union.

Subset and set difference query. For a subset query (positive or negative), we use the property $S_i \subseteq S_j \Leftrightarrow \forall y \in S_i : y \in S_j$. For a set difference query we use the property $\mathsf{D} = S_i - S_j \Leftrightarrow \mathsf{D} \subseteq S_i \wedge S_i - \mathsf{D} = S_i \cap S_j$. The details of the algorithms concerning these queries can be found in Section 9 in the Appendix. We now give the final theorem of our work. Its proof is in the Appendix, including detailed proof of security.

Theorem 1 *Consider a collection of m sets $S_1, \ldots, S_m$ and let $M = \sum_{i=1}^{m} |S_i|$ and $0 < \epsilon < 1$. For a query operation involving t sets, let N be the sum of the sizes of the involved sets, and δ be the answer size. There exists an authenticated data structure scheme $\mathcal{ASC} = \{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$ for a sets collection data structure D with the following properties: (1) it is correct and secure according to Definitions 2 and 3 and based on the bilinear q-strong Diffie-Hellman assumption; (2) The access complexity of algorithm (i) $\mathsf{genkey}()$ is $O(1)$; (ii) $\mathsf{setup}()$ is $O(m + M)$; (iii) $\mathsf{update}()$ is $O(1)$ outputting information upd of $O(1)$ group complexity; (iv) $\mathsf{refresh}()$ is $O(1)$; (3) For all queries q (intersection/union/subset/difference), algorithm $\mathsf{query}()$ has $O(N \log^2 N \log\log N + tm^{\epsilon} \log m)$ access complexity, algorithm $\mathsf{verify}()$ has $O(t + \delta)$ access complexity and the proof $\Pi(q)$ has $O(t + \delta)$ group complexity; (4) The group complexity of the authenticated data structure $\mathsf{auth}(D)$ is $O(m + M)$.*

4 Security, protocols and applications

In this section we give a proof sketch for the security (based on Definition 3) of one query, i.e., that of the intersection and comment on how our authenticated data structure scheme can be used by a three-party protocol [39] and a two-party protocol [33]. We also talk about some applications.

Security proof sketch. Let D_0 be a sets collection data structure consisting of m sets $S_1, S_2, \ldots, S_m$ and let $\mathcal{ASC} = \{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$ be our authenticated data structure scheme. Let k be the security parameter and let $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^k)$. The adversary is given the public key pk, namely the values $\{h(.), (p, \mathbb{G}, \mathcal{G}, e, g), g^s, g^{s^2}, \ldots, g^{s^q}\}$ and unlimited access to all the algorithms of $\mathcal{ASC}$, except for $\mathsf{setup}()$ and $\mathsf{update}()$ to which he only has oracle access. The adversary initially outputs the authenticated data structure $\mathsf{auth}(D_0)$ and the digest d_0, through an oracle call to algorithm $\mathsf{setup}()$. Then the adversary picks a polynomial number of updates u_i (e.g., insert an element x into a set S_r) and outputs the data structure D_i, the authenticated data structure $\mathsf{auth}(D_i)$ and the digest d_i through oracle access to $\mathsf{update}()$. Then he picks a set of indices $q = \{1, 2, \ldots, t\}$ (wlog), all between 1 and m and outputs a proof $\Pi(q)$ and an answer $\mathcal{I} \neq \mathsf{I} = S_1 \cap S_2 \cap \ldots \cap S_t$ which is rejected by $\mathsf{check}()$ (incorrect). Suppose the answer $\alpha(q)$ contains d elements. The proof $\Pi(q)$ contains (i) *Some* coefficients $\beta_0, \beta_1, \ldots, \beta_d$; (ii) *Some* accumulation values acc_j with *some* respective proofs Π_j, for $j = 1, \ldots, t$; (iii) *Some* subset witnesses W_j with *some* body witnesses F_j, for $j = 1, \ldots, t$ (this is what algorithm $\mathsf{verify}()$ expects for input). Suppose $\mathsf{verify}()$ accepts. Then: (a) By Lemma 3, $\beta_0, \beta_1, \ldots, \beta_d$ are *indeed* the coefficients of the polynomial $\prod_{x \in \mathcal{I}}(x + s)$, except with negligible probability; (b) By Lemma 5, values acc_j are *indeed* the accumulation values of sets S_j, except with negligible probability; (c) By Lemma 1, values W_j are *indeed* the subset witnesses for set $\mathcal{I}$ (with reference to S_j), i.e., $\mathsf{W}_j = g^{P_j(s)}$, except with negligible probability; (d) However, $P_1(s), P_2(s), \ldots, P_t(s)$ are not coprime since $\mathcal{I}$ is incorrect and therefore $\mathcal{I}$ cannot contain *all* the elements. By the verification of Relation 9 (completeness condition), we can derive a polynomial-time algorithm that outputs a bilinear q-strong Diffie-Hellman challenge $(a, e(g, g)^{1/(s+a)})$ for an element a that is a common factor of the polynomials $P_1(s), P_2(s), \ldots, P_t(s)$. The full proof is in the Appendix.

Protocols. As we mentioned in the introduction, an authenticated data structure scheme $\mathcal{A}$ may be used by a three-party protocol [39]: A trusted entity, called *source*, owns a data structure D_h, but desires to outsource query answering, in a trustworthy (verifiable) way. The source runs $\mathsf{genkey}()$ and $\mathsf{setup}()$, outputs the authenticated data structure $\mathsf{auth}(D_h)$ along with the digest d_h. The source subsequently signs the digest d_h, and it outsources $\mathsf{auth}(D_h)$, D_h, the digest d_h and its signature (which is forwarded to the *clients* during verification) to some untrusted entities, called *servers*. On input a data structure query q sent by the clients, the servers use $\mathsf{auth}(D_h)$ and D_h to compute proofs $\Pi(q)$, by running algorithm $\mathsf{query}()$. Clients can verify these proofs $\Pi(q)$ by running algorithm $\mathsf{verify}()$, and since they have access to the signature of d_h. When there is an update in the data structure—issued by the source—, the source uses algorithm $\mathsf{update}()$ to produce the new digest d'_h to be used for the next verification, while the servers update the authenticated data structure through $\mathsf{refresh}()$. The corollary below summarizes the three-party protocol:

Corollary 1 (Three-party protocol) *Let k be the security parameter, $S_1, \ldots, S_m$ be a sets collection data structure D, $M = \sum_{i=1}^{m} |S_i|$, $0 < \epsilon < 1$ and $\mathcal{ASC}$ be a correct and secure authenticated data structure scheme for D. For a query operation involving t sets, let N be the sum of the sizes of the involved sets and δ be the answer size. There exists a three-party authenticated data structures protocol involving a trusted source, an untrusted server and a client for verifying queries on D such that: (a) The setup at the source has $O(m + M)$ access complexity; (b) The update at the source has $O(1)$ access complexity; (c) The space needed at the source and the server has $O(m + M)$ group complexity; (d) The communication between the source and the server has $O(1)$ group complexity; (e) The update at the server has $O(1)$ access complexity; (f) For all queries q (intersection/union/subset/set difference), the query at the server has $O(N \log^2 N \log\log N + tm^{\epsilon} \log m)$ access complexity, the verification at the client has $O(t + \delta)$ access complexity and the proof $\Pi(q)$ has $O(t + \delta)$ group complexity; (g) For a query q sent by the client to the server at any time (even after updates), let α be an answer and let π be a proof returned by the server. With probability $\Omega(1 - \mathsf{neg}(k))$, the client accepts the answer α if and only if α is correct.*

We note here that an authenticated data structure scheme $\mathcal{A}$ can also be used by a *non-interactive* two-party authenticated data structures protocol [33]: In this case, the *source* and the *client* coincide (i.e., the source issues both the updates and the queries) and the source is required to keep only constant state, i.e., the digest. A non-interactive two-party protocol that uses an authenticated data structure scheme for a data structure D is *directly* comparable with the recently appeared model of *outsourced verifiable computation* [2, 11, 17] for the functionalities offered by the data structure D, e.g., computation of intersection, union, etc.:

Corollary 2 (Two-party protocol) *Let k be the security parameter, $S_1, \ldots, S_m$ be a sets collection data structure D, $M = \sum_{i=1}^{m} |S_i|$, $0 < \epsilon < 1$ and $\mathcal{ASC}$ be a correct and secure authenticated data structure scheme for D. For a query operation involving t sets, let N be the sum of the sizes of the involved sets and δ be the answer size. There exists a non-interactive two-party authenticated data structures protocol involving a trusted source and an untrusted server for verifying queries on D such that: (a) The setup at the source has $O(m+M)$ access complexity; (b) The update at the source has $O(1)$ access complexity; (c) The space needed at the source has $O(1)$ group complexity while the space needed at the server has $O(m+M)$ group complexity; (d) The update at the server has $O(m^{\epsilon} \log m)$ access complexity; (e) For all queries q (intersection/union/subset/set difference), the query at the server has $O(N \log^2 N \log\log N + tm^{\epsilon} \log m)$ access complexity, the verification at the client has $O(t + \delta)$ access complexity and the proof $\Pi(q)$ has $O(t + \delta)$ group complexity; (f) For a query q sent by the source to the server at any time (even after updates), let α be an answer and let π be a proof returned by the server. With probability $\Omega(1 - \mathsf{neg}(k))$, the source accepts the answer α if and only if α is correct.*

Applications. First of all, our scheme can be used to verify *keyword-search queries* implemented by the *inverted index* data structure [5]: Each term in the dictionary corresponds to a set in our sets collection data structure which contains all the documents that include this term. A usual text query for terms m_1 and m_2 returns those documents that are included in both the sets that are represented by m_1 and m_2, i.e., their intersection. Moreover, the derived *authenticated inverted index* can be efficiently updated as well. However, sometimes in keyword searches (e.g., keyword searches in the email inbox) it is desirable to introduce a "second" dimension: For example, a query could be "give me the emails that contain terms m_1 and m_2 and which were received between time t_1 and t_2", where $t_1 < t_2$. We call this procedure timestamped keyword-search. One solution for that could be to embed a timestamp in the documents (e.g., each email message) and have the client do the filtering locally, after he has verified—using our scheme—the intersection of the sets that correspond to terms m_1 and m_2. However, this is not *operation-sensitive* at all: The intersection can be a lot bigger than the set resulted after the application of the local filtering, making this straightforward solution inefficient. By using a *segment-tree* data structure [38], we show in the Appendix how this problem is solved efficiently (see Theorem 2). Finally, in Theorem 3 in the Appendix we show how to use our method for the verification of equi-join queries over relational tables. Experimental

results in the Appendix (Section 12) indicate that our schemes compare favorably with existing work.

Conclusions and open problems. In this paper, we presented an authenticated data structure for the optimal verification of set operations. The achieved efficiency is mainly due to new, extended security properties of accumulators based on pairing-based cryptography. Our solution provides two important properties, namely *public verifiability* and *efficiency*, as opposed to the outsourced verifiable computation model. A natural question to ask is whether outsourced verifiable computation with *secrecy*, *public verifiability* and *efficiency* exists. Analogously, which other specific functionalities can be optimally and publicly verified? Finally, according to a recently proposed definition of optimality [36], our construction is *nearly* optimal—only *verification costs* (hence the title) and updates are optimal. It is interesting to explore whether an *optimal* authenticated sets collection data structure is possible, i.e., one that asymptotically matches the bounds of the plain sets collection data structure, reducing the query time from $O(N \log^2 N)$ to $O(N)$.

References

[1] PBC: The pairing-based cryptography library. http://crypto.stanford.edu/pbc/.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 152–163. Springer, 2010.

[3] M. J. Atallah, Y. Cho, and A. Kundu. Efficient data authentication in an environment of untrusted third-party distributors. In *Proceedings of International Conference on Data Engineering (ICDE)*, 2008.

[4] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *Proc. Cryptographers' Track at the RSA Conference (CT-RSA)*, pages 295–308. Springer, 2009.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Publishing Company, Reading, Mass., 1999.

[6] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT, LNCS 1233*, pages 163–192. Springer-Verlag, 1997.

[7] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[8] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[9] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[10] S. Canard and A. Gouget. Multiple denominations in e-cash with compact transaction data. In *Financial Cryptography*, pages 82–97, 2010.

[11] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, 2010.

[12] I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. http://eprint.iacr.org/.

[13] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.

[14] C. Dwork, M. Naor, G. N. Rothblum, and V. Vaikuntanathan. How efficient can memory checking be? In *Theoretical Cryptography Conference (TCC)*, pages 503–520, 2009.

[15] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[16] J. V. Z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[17] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[18] M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. Information Security Conference (ISC)*, volume 2433 of *LNCS*, pages 372–388. Springer, 2002.

[19] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. DARPA Information Survivability Conference and Exposition II (DISCEX II)*, pages 68–82, 2001.

[20] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *Proc. RSA Conference, Cryptographers' Track (CT-RSA)*, volume 4964 of *LNCS*, pages 407–424. Springer, 2008.

[21] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Efficient authenticated data structures for graph connectivity and geometric search problems. *Algorithmica*, 2009. Online First.

[22] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. In *SODA*, pages 158–167. SIAM, 2003.

[23] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *Proc. Applied Cryptography and Network Security (ACNS)*, pages 253–269, 2007.

[24] B. Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University, November 2008.

[25] K. M. Man Lung Yiu, Yimin Lin. Efficient verification of shortest path search via authenticated hints. In *ICDE*, pages 237–248, 2010.

[26] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[27] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1989.

[28] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2213–2218, 2003.

[29] R. Morselli, S. Bhattacharjee, J. Katz, and P. J. Keleher. Trust-preserving set operations. In *INFOCOM*, 2004.

[30] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.

[31] L. Nguyen. Accumulators from bilinear pairings and applications. In *Proc. RSA Conference, Cryptographers' Track (CT-RSA) , LNCS 3376, pp. 275-292, Springer.*, 2005.

[32] H. Pang and K.-L. Tan. Authenticating query results in edge computing. In *Proc. Int. Conf. on Data Engineering*, pages 560–571, 2004.

[33] C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In *Proc. Int. Conference on Information and Communications Security (ICICS)*, volume 4861 of *LNCS*, pages 1–15. Springer, 2007.

[34] C. Papamanthou and R. Tamassia. Update-optimal authenticated structures based on lattices. Cryptology ePrint Archive, Report 2010/128, 2010. `http://eprint.iacr.org/`.

[35] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 437–448. ACM, October 2008.

[36] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal authenticated data structures with multilinear forms. In *Proc. Int. Conference on Pairing-Based Cryptography (PAIRING)*, pages 246–264, 2010.

[37] F. P. Preparata and D. V. Sarwate. Computational complexity of Fourier transforms over finite fields. *Mathematics of Computation*, 31(139):pp. 740–751, 1977.

[38] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[39] R. Tamassia. Authenticated data structures. In *Proc. European Symp. on Algorithms*, volume 2832 of *LNCS*, pages 2–5. Springer-Verlag, 2003.

[40] R. Tamassia and N. Triandopoulos. Certification and authentication of data structures. In *Proc. Alberto Mendelzon Workshop on Foundations of Data Management*, 2010.

[41] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated join processing in outsourced databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 5–18, 2009.

Appendix

5 Proof of Lemma 1

Suppose there is a polynomial-time algorithm that computes such a set $\mathcal{S} = \{y_1, y_2, \ldots, y_\ell\}$. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and $y_j \notin \mathcal{X}$ for some $1 \leq j \leq \ell$. That means that

$$e(\mathsf{W}, g)^{\prod_{y \in \mathcal{S}}(y+s)} = e(g,g)^{(x_1+s)(x_2+s)\ldots(x_n+s)}\,.$$

Note that $(y_j + s)$ does not divide $(x_1 + s)(x_2 + s)\ldots(x_n + s)$. Therefore there exist polynomial $Q(s)$ of degree $n-1$ and constant λ, such that $(x_1 + s)(x_2 + s)\ldots(x_n + s) = Q(s)(y_j + s) + \lambda$. Thus we have

$$\begin{aligned} e(\mathsf{W}, g)^{(y_j+s)\prod_{1\leq i\neq j\leq \ell}(y_i+s)} &= e(g,g)^{Q(s)(y_j+s)+\lambda} \Rightarrow \\ e(g,g)^{\frac{1}{y_j+s}} &= \left[e(\mathsf{W}, g)^{\prod_{1\leq i\neq j\leq \ell}(y_i+s)} e(g,g)^{-Q(s)}\right]^{\lambda^{-1}}\,. \end{aligned}$$

This means that the algorithm can be used to break the bilinear q-strong Diffie-Hellmann assumption.

6 Proof of Lemma 3

The complexity of algorithm $\mathsf{certify}()$ has access complexity $O(n)$ since it involves $O(n)$ multiplications, additions and exponentiations. The probability that $\mathsf{certify}()$ accepts while $a_0, a_1, \ldots, a_n$ are not the coefficients of the polynomial that has roots $-x_1, -x_2, \ldots, -x_n$ is equal to the probability of κ being the root of the polynomial $R(\kappa) = \sum_{i=0}^{n} a_i\kappa^i - \prod_{i=1}^{m}(\kappa + x_i)$. This follows from polynomial equality that should hold for all κ. Now, polynomial $R(\kappa)$ has degree $n = \mathsf{poly}(k)$ and has $O(n)$ roots. Since κ is picked at random from $\mathbb{Z}_p^*$, it follows that this probability is bounded by $O(\mathsf{poly}(k)/2^k)$, which is $\mathsf{neg}(k)$, and therefore the validity of the coefficients can be verified with probability $\Omega(1 - \mathsf{neg}(k))$ with $\Theta(n)$ complexity.

7 Proof of Lemma 5

Let $1 \leq i \leq m$ and let $v_0, v_1, \ldots, v_l$ be a path in the tree. The adversary Adv outputs an incorrect answer $\alpha_i \neq g^{\prod_{x\in S_i}(x+s)} = \mathsf{acc}(S_i)$ and also a proof $\Pi_i = (\pi_1, \pi_2, \ldots, \pi_l)$ ($l = \lceil \frac{1}{\epsilon} \rceil$) where $\pi_j = (\beta_j, \gamma_j)$ (see algorithm $\mathsf{queryTree}()$). We define now the following events, related to the choice of the proof above made by the adversary. Our goal will be to the express the probability that $\mathsf{verifyTree}(i, \alpha_i, \Pi_i, d_h, \mathsf{pk})$ accepts and $\alpha_i \neq \mathsf{acc}(S_i)$ as a function of the following events. Note that d_h is the correct digest of the authenticated data structure:

1. $E_{0,0}$: The value α_i picked by Adv is such that $\alpha_i \neq \mathsf{acc}(S_i)$.
2. E_1: The value β_1 picked by Adv is such that $e(\beta_1, g) = e(\alpha_i, g^s g^i)$. This event can be partitioned into two mutually exclusive events, i.e., $E_1 = E_{1,0} \cup E_{1,1}$ such that
   - $E_{1,0}$: Value β_1 is *not* a correctly formed digest of node v_0, as defined in Relation 3;
   - $E_{1,1}$: Value β_1 is a correctly formed digest of node v_0, as defined in Relation 3, i.e., it is

     $$\beta_1 = g^{(s+\ell)\prod_{x\in S_\ell}(s+x)}\,,$$

     for some set S_ℓ and where $1 \leq \ell \leq m$.
3. E_j: For $j = 2, \ldots, \ell$, the values β_j, β_{j-1} and γ_{j-1} picked by Adv are such that

   $$e\left(\beta_j, g\right) = e\left(\gamma_{j-1}, g^s g^{h(\beta_{j-1})}\right) \text{ for all } 2 \leq j \leq l\,.$$

   This event can be partitioned into two mutually exclusive events, i.e., $E_j = E_{j,0} \cup E_{j,1}$ such that
   - $E_{j,0}$: Value β_j is *not* a correctly formed digest of node v_{j-1}, as defined in Relation 3;

- $E_{j,1}$: Value β_j is the is a correctly formed digest of node v_{j-1}, as defined in Relation 3.

4. $E_{\ell,1}$: The values β_l and γ_l picked by Adv are such that

$$e\left(d_h, g\right) = e\left(\gamma_l, g^s g^{h(\beta_l)}\right).$$

The probability that verifyTree() accepts, while $\alpha_i \neq \mathsf{acc}(S_i)$ is the probability

$$\begin{aligned}
\Pr[E_{0,0} \cap E_1 \cap E_2 \cap \ldots \cap E_{l,1}] &= \Pr[E_{0,0} \cap (E_{1,0} \cup E_{1,1}) \cap (E_{2,0} \cup E_{2,1}) \cap \ldots \cap E_{l,1}] \\
&\leq \Pr[E_{0,0}|E_{1,1}] + \Pr[E_{1,0}|E_{2,1}] + \Pr[E_{2,0}|E_{3,1}] + \ldots + \Pr[E_{l-1,0}|E_{l,1}] \\
&= \Pr[E_{0,0}|E_{1,1}] + \sum_{j=2}^{l} \Pr[E_{j-1,0}|E_{j,1}].
\end{aligned}$$

First we examine the event $E_{0,0}|E_{1,1}$. The event $E_{1,1}$ implies that the adversary has found a value α_i such that

$$e\left(g^{(s+\ell)\prod_{x \in S_\ell}(s+x)}, g\right) = e(\alpha_i, g^s g^i).$$

By Lemma 1, with probability $\Omega(1 - \mathsf{neg}(k))$, it has to be $i \in \{\ell\} \cup S_j$. However, since $1 \leq \ell \leq m$ and $1 \leq i \leq m$ and for all $x \in S_\ell$ it is $x > m$, we conclude that that $i = \ell$. Therefore $\alpha_i = \mathsf{acc}(S_i)$, with probability $\Omega(1 - \mathsf{neg}(k))$. As such $\Pr[E_{0,0}|E_{1,1}] \leq \mathsf{neg}(k)$.

For the remaining events $E_{j-1,0}|E_{j,1}$ ($j \geq 2$), note that by the one-to-one property of the function $h(.)$, $E_{j-1,0}$ is equivalent with the event "value $h(\beta_{j-1})$ is *not* the $h(.)$ value of the digest of node v_{j-2}, as defined in Relation 3". However, by the definition of the digest β_j (the digest of the father of v_{j-2} is the accumulation value of the $h(.)$ values of the digests of all of it children, see Relation 3), the event $E_{j-1,0}|E_{j,1}$ implies breaking the bilinear q-strong Diffie-Hellman assumption (Assumption 1), by Lemma 1. Therefore for all $j = 1, \ldots, l$, $\Pr[E_{j-1,0}|E_{j,1}]$ is $\mathsf{neg}(k)$. Since $l = O(1)$, the total probability is also $\mathsf{neg}(k)$. This concludes the proof.

8 Proof of Lemma 6

($\Rightarrow$) This direction follows by the fact that we can use the extended Euclidean algorithm and find polynomials $q_1(s), \ldots, q_t(s)$ such that

$$q_1(s)P_1(s) + \ldots + q_t(s)P_t(s) = \mathsf{GCD}(P_1(s), P_2(s), \ldots, P_t(s)).$$

Since $P_1(s), P_2(s), \ldots, P_t(s)$ share no common factors , it follows that $\mathsf{GCD}(P_1(s), P_2(s), \ldots, P_t(s)) = 1$.

($\Leftarrow$) Suppose there exist polynomials $q_1(s), q_2(s), \ldots, q_t(s)$ that satisfy relation $q_1(s)P_1(s) + q_2(s)P_2(s) + \ldots + q_t(s)P_t(s) = 1$ but I is not the intersection. This means that polynomials $P_1(s), P_2(s), \ldots, P_t(s)$ share at least one common factor, e.g., $(s + r)$. Therefore there exists some polynomial $A(s)$ such that $(s + r)A(s) = 1$, i.e., the polynomials $(s + r)A(s)$ and 1 are *equal*, which is a contradiction (note that we want the polynomials to be equal for every $s \in \mathbb{Z}_p$).

In order to compute these coefficients, we use the extended Euclidean algorithm recursively, based on the fact that the greatest common divisor $\mathsf{GCD}(P_1(s), \ldots, P_t(s))$ equals $\mathsf{GCD}(P_1(s), \mathsf{GCD}(P_2(s), \ldots, P_t(s)))$. To compute the greatest common divisor of two $O(n)$-degree polynomials, we can use the algorithm described in [16] that has $O(n \log^2 n \log\log n)$ complexity. Since we are using this algorithm t times, the time complexity is $O(tn \log^2 n \log\log n)$. Moreover, by the property that $x \log x + y \log y \leq (x + y) \log(x + y)$ and since the size of the sets participating in the intersection is N this equals $O(N \log^2 N \log\log N)$. This algorithm also outputs the required coefficients. If we arrange our data (i.e., t polynomials) on a binary tree, after all the coefficients of the internal nodes have been computed, the final coefficients for all elements at the leaves can be computed in $O(t)$ multiplications (we can avoid the $O(t \log t)$ cost) of $O(n_i)$ degree polynomials, where n_i are the degrees of the polynomials of the leaves. Therefore the result holds.

9 Subset and set difference query

Subset. Here we use the property $S_i \subseteq S_j \Leftrightarrow \forall y \in S_i : y \in S_j$.

Algorithm $\{\Pi(q), \alpha(q)\} \leftarrow \mathsf{query}(q, D_h, \mathsf{auth}(D_h), \mathsf{pk})$**:** (*subset*)
The proof contains the following elements: **(1)** Accumulation values $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$ associated with the sets S_i and S_j, along with their respective proofs Π_i and Π_j, output by $\mathsf{queryTree}(i, D_h, \mathsf{auth}(D_h), \mathsf{pk})$ and $\mathsf{queryTree}(j, D_h, \mathsf{auth}(D_h), \mathsf{pk})$ ($O(m^\epsilon m)$ complexity by Lemma 4); **(2)** For a positive answer (i.e., $S_i \subseteq S_j$), the proof contains a subset witness of S_i with respect to S_j, i.e., the value W_{S_i,S_j} as defined in Relation 1 ($O(N \log N)$ complexity by Lemma 2). For a negative answer ($S_i \nsubseteq S_j$), the algorithm picks a $y \in S_i$ such that $y \notin S_j$, returns a membership proof of y in S_i, i.e., the witness W_{y,S_i} as defined in Relation 1. Also it returns a proof for an *empty* intersection of the sets $\{y\}$ and S_j, i.e., the *completeness witnesses* $\mathsf{F}_y = g^{q_y(s)}$ and $\mathsf{F}_j = g^{q_j(s)}$ such that $(y+s)q_y(s) + P_j(s)q_j(s) = 1$[13] (only the proof for the completeness condition is needed, i.e., Relation 9). The total complexity is $O(N \log^2 N \log\log N + m^\epsilon \log m)$ (Lemmata 4 and 6) and the group complexity of the proof is $O(1)$.

Algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \alpha(q), \Pi(q), d_h, \mathsf{pk})$**:** (*subset*)
The verification for query "is $S_i \subseteq S_j$?" is as follows: **(1)** Verify the integrity of $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$ by using $\mathsf{verifyTree}(i, \mathsf{acc}(S_i), \Pi_i, d_h, \mathsf{pk})$ and $\mathsf{verifyTree}(j, \mathsf{acc}(S_j), \Pi_j, d_h, \mathsf{pk})$ with $O(1)$ complexity, by Lemma 4 (else output reject); **(2)** For a positive answer, output reject if $e(\mathsf{W}_{S_i,S_j}, \mathsf{acc}(S_i)) \neq e(\mathsf{acc}(S_j), g)$. Otherwise output reject if either (a) $e(\mathsf{W}_{y,S_i}, g^y g^s) \neq e(\mathsf{acc}(S_i), g)$ or (b) $e(g^y g^s, \mathsf{F}_y) e(\mathsf{acc}(S_j), \mathsf{F}_j) \neq e(g, g)$. Else output accept. The whole test has $O(1)$ complexity.

Set difference query. Let $\mathsf{D} = S_i - S_j$. It is $\mathsf{D} = S_i - S_j \Leftrightarrow \mathsf{D} \subseteq S_i \wedge S_i - \mathsf{D} = S_i \cap S_j$.

Algorithm $\{\Pi(q), \alpha(q)\} \leftarrow \mathsf{query}(q, D_h, \mathsf{auth}(D_h), \mathsf{pk})$**:** (*set difference*)
The proof contains the following pieces: The query q is the set of indices $\{i, j\}$, requiring the difference $\mathsf{D} = S_i - S_j$. Let $\alpha(q) = \{y_1, y_2, \ldots, y_\delta\}$ be the difference D. The proof $\Pi(q)$ consists of the following parts: **(1)** *Coefficients* $b_\delta, b_{\delta-1}, \ldots, b_0$ of polynomial $(s+y_1)(s+y_2)\ldots(s+y_\delta)$ associated with the difference $\mathsf{D} = \{y_1, y_2, \ldots, y_\delta\}$, computed with $O(\delta \log \delta)$ complexity (Lemma 2), and of total group complexity $O(\delta)$; **(2)** *Accumulation values* $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$ associated with the sets S_i and S_j, along with their respective proofs Π_i and Π_j, output by calling algorithm $\mathsf{queryTree}(i, D_h, \mathsf{auth}(D_h), \mathsf{pk})$ and $\mathsf{queryTree}(j, D_h, \mathsf{auth}(D_h), \mathsf{pk})$, the computation of which has $O(m^\epsilon \log m)$ complexity (Lemma 4). Moreover the group complexity of both Π_i and Π_j is $O(1)$ (Lemma 4); **(3)** A subset witness $\mathsf{W}_{\mathsf{D},S_i}$; **(4)** A proof that $S_i - \mathsf{D}$ is the intersection of S_i and S_j, i.e., the values $(\mathsf{W}_{\mathsf{I},i}, \mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},i}, \mathsf{F}_{\mathsf{I},j})$ for the intersection $\mathsf{I} = S_i \cap S_j$, as computed before. Note that the intersection itself is not returned. Summing up, contructing the proof has $O(N \log^2 N \log\log N + m^\epsilon \log^2 m)$ complexity (Lemmata 4 and 6) and the proof has $O(\delta)$ group complexity.

Algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \alpha(q), \Pi(q), d_h, \mathsf{pk})$**:** (*set difference*)
The verification algorithm performs the following steps: **(1)** First it uses $\mathbf{b} = [b_\delta, b_{\delta-1}, \ldots, b_0]$ and the answer $\alpha(q) = \{y_1, y_2, \ldots, y_\delta\}$, and calls $\mathsf{certify}(\mathbf{b}, \alpha(q), \mathsf{pk})$ in order to certify the validity of $b_\delta, b_{\delta-1}, \ldots, b_0$. This step, by Lemma 3, has $O(\delta)$ complexity. If $\mathsf{certify}()$ rejects, the algorithm outputs reject; **(2)** The algorithm verifies the integrity of $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$ by using the algorithm $\mathsf{verifyTree}(i, \mathsf{acc}(S_i), \Pi_i, d_h, \mathsf{pk})$ and the algorithm $\mathsf{verifyTree}(j, \mathsf{acc}(S_j), \Pi_j, d_h, \mathsf{pk})$ with $O(1)$ complexity, by Lemma 4 (else output reject); **(3)** The algorithm outputs reject if either (a) $e(\mathsf{W}_{\mathsf{D},S_i}, \mathsf{acc}(\mathsf{D})) \neq e(\mathsf{acc}(S_i), g)$; or (b) The completeness proof $(\mathsf{W}_{\mathsf{I},i}, \mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},i}, \mathsf{F}_{\mathsf{I},j})$ for the intersection $\mathsf{I} = S_i \cap S_j$ does not verify (note we are not using the explicit intersection elements in this verification, but $\mathsf{W}_{\mathsf{D},S_i}$. Summing up, the algorithm has $O(\delta)$ complexity.

[13] We recall $P_j(s) = \prod_{x \in S_j}(x+s)$.

10 Proof of Theorem 1

10.1 Correctness

Let D_0 be any sets collection data structure containing m sets. Fix the security parameter k and output $\mathsf{pk} = \{h(.), (p, \mathbb{G}, \mathcal{G}, e, g), g^s, g^{s^2}, \ldots, g^{s^q}\}$ and $\mathsf{sk} = s$ by calling algorithm $\mathsf{genkey}()$. Then output an authenticated data structure $\mathsf{auth}(D_0)$ and the respective digest d_0, by calling algorithm $\mathsf{setup}()$. Pick a polynomial number of updates—namely, pick a polynomial number of elements for insertion (or deletion) into (or from) a set S_r—and update $\mathsf{auth}(D_0)$ and d_0 by calling algorithm $\mathsf{refresh}()$. Let D_h be the final sets collection data structure, $\mathsf{auth}(D_h)$ be the produced authenticated data structure and d_h be the final digest. By the way $\mathsf{refresh}()$ works (see Relation 4), at every time, the digest $d(v)$ of a node v of the tree T is maintaining the following property: If v_i is a leaf corresponding to number $1 \leq i \leq m$ (i.e., to set S_i), it always is $d(v_i) = \mathsf{acc}(S_i)^{(s+i)}$. Otherwise, for every other node v of T, it always is

$$d(v) = g^{\prod_{w \in \mathcal{N}(v)} (s + h(d(w)))} ,$$

where $\mathcal{N}(v)$ denotes the set of children of node v. We prove correctness for all four query operations, i.e., for intersection, union, subset and difference.

Intersection. Let our query q be $\{1, 2, \ldots, t\}$ (wlog), i.e., a set of indices that refer to the intersection of sets $S_1, S_2, \ldots, S_t$. Algorithm $\mathsf{query}()$ outputs the proof $\Pi(q)$ and the *correct* answer $\mathsf{I} = \{y_1, y_2, \ldots, y_\delta\} = S_1 \cap S_2 \cap \ldots \cap S_t$. The proof $\Pi(q)$ for the intersection contains the following parts:

1. The coefficients $b_\delta, b_{\delta-1}, \ldots, b_0$ of polynomial $(s + y_1)(s + y_2) \ldots (s + y_\delta)$ associated with the intersection $\mathsf{I} = \{y_1, y_2, \ldots, y_\delta\}$. Since for every $\kappa \in \mathbb{Z}_p$ it is $\sum_{i=0}^{\delta} b_i \kappa^i = \prod_{i=1}^{m} (\kappa + y_i)$, Algorithm $\mathsf{certify}()$ accepts;
2. The accumulation values $\mathsf{acc}(S_j) = g^{\prod_{x \in S_j} (x+s)}$ associated with the sets S_j, using their respective proofs Π_j, output by $\mathsf{queryTree}(j, D_h, \mathsf{auth}(D_h), \mathsf{pk})$, for $j = 1, \ldots, t$. We recall that each Π_j is the ordered sequence $\pi_1, \pi_2, \ldots, \pi_l$ ($l = \lceil 1/\epsilon \rceil$), where π_i is the tuple $(d(v_{i-1}), \mathsf{W}_{v_{i-1}(v_i)})$, where

   $$\mathsf{W}_{v_{i-1}(v_i)} = g^{\prod_{w \in \mathcal{N}(v_i) - \{v_{i-1}\}} (s + h(d(w)))} ,$$

   is the witness for a subset of *one* element (i.e., the element $h(d(v_{i-1}))$), as defined in Section 2. By Relation 3, $\mathsf{verifyTree}(j, \mathsf{acc}(S_j), \Pi_j, d_h, \mathsf{pk})$, on such inputs, always accepts;
3. The subset witnesses $\mathsf{W}_{\mathsf{I},j} = g^{P_j(s)} = g^{\prod_{x \in S_j - \mathsf{I}} (x+s)}$ for $j = 1, \ldots, t$. It is $e(\mathsf{W}_{\mathsf{I},j}, \mathsf{acc}(\mathsf{I})) = e(\mathsf{acc}(S_j), g)$, by the properties of the bilinear map, therefore Algorithm $\mathsf{verify}()$ does not reject in this item either;
4. The completeness witnesses $\mathsf{F}_{\mathsf{I},j} = g^{q_j(s)}$ for $j = 1, \ldots, t$. It is

   $$\prod_{j=1}^{t} e\left(\mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},j}\right) = e(g, g)^{\sum_{j=1}^{t} q_j(s) P_j(s)} = e(g, g) ,$$

   since by the construction of the completeness witnesses it should be $\sum_{j=1}^{t} q_j(s) P_j(s) = 1$.

This completes the proof of correctness for the case of intersection, since we proved that for every intersection query q and for every correct answer and proof output by $\mathsf{query}()$, $\mathsf{verify}()$ always accepts.

Union. Let our query q be $\{1, 2, \ldots, t\}$ (wlog), i.e., a set of indices that refer to the union of sets $S_1, S_2, \ldots, S_t$. Algorithm $\mathsf{query}()$ outputs the proof $\Pi(q)$ and the *correct* answer $\mathsf{U} = \{y_1, y_2, \ldots, y_\delta\} = S_1 \cup S_2 \cup \ldots \cup S_t$. The proof $\Pi(q)$ for a union contains the following parts:

1. The coefficients $b_\delta, b_{\delta-1}, \ldots, b_0$ and the accumulation values $\mathsf{acc}(S_j) = g^{\prod_{x \in S_j} (x+s)}$. These are always verified as in the case of intersection. See Items 1 and 2 above;

2. The membership witnesses W_{y_i,S_k} for some $k = 1,\ldots,t$, for each element y_i ($i = 1,\ldots,\delta$). For $i = 1,\ldots,\delta$, it is $e(\mathsf{W}_{y_i,S_k}, g^{y_i}g^s) = e(\mathsf{acc}(S_k), g)$, since W_{y_i,S_k} is the subset witness as defined in Relation 1;
3. The subset witnesses $\mathsf{W}_{S_j,\mathsf{U}}$, for all $j = 1,\ldots,t$. For all $j = 1,\ldots,t$ it is

$$e(\mathsf{W}_{S_j,\mathsf{U}}, \mathsf{acc}(S_j)) = e(\mathsf{acc}(\mathsf{U}), g)\,,$$

where $\mathsf{W}_{S_j,\mathsf{U}}$ is the subset witness of S_j with respect to U, as defined in Relation 1 and $\mathsf{U} \supseteq S_j$ for all $j = 1,\ldots,t$;

This completes the proof of correctness for the case of union, since we proved that for every union query q and for every correct answer and proof output by query(), verify() always accepts.

Subset. Let our query be "is $S_i \subseteq S_j$?. Algorithm query() outputs the proof $\Pi(q)$ and the *correct* answer, i.e., either true or false. The proof $\Pi(q)$ for a subset query contains the following parts:

1. The accumulation values $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$. These are always verified as in the case of intersection. See Item 2 above;
2. Depending on whether we have a positive or a negative answer, we distinguish the following cases:
   - Positive answer, i.e., S_i is a subset of S_j. The proof contains the subset witness W_{S_i,S_j}. Then it is $e(\mathsf{W}_{S_i,S_j}, \mathsf{acc}(S_i)) = e(\mathsf{acc}(S_j), g)$, by the definition of W_{S_i,S_j} (see Relation 1) and since $S_i \subseteq S_j$;
   - Negative answer, i.e., S_i is not a subset of S_j. The proof contains an element y such that $y \in S_i$ but $y \notin S_j$, the respective membership witness W_{y,S_i} and two completeness witnesses F_y and F_j. It is $e(\mathsf{W}_{y,S_i}, g^y g^s) = e(\mathsf{acc}(S_i), g)$, by definition of W_{y,S_i} in Relation 1 and since $y \in S_i$. Also it holds $e(g^y g^s, \mathsf{F}_y)e(\mathsf{acc}(S_j), \mathsf{F}_j) = e(g, g)$, since $\mathsf{F}_y = g^{q_y(s)}$ and $\mathsf{F}_j = g^{q_j(s)}$ are such that $(y+s)q_y(s) + q_j(s)\prod_{x\in S_j}(x+s) = 1$ and $y \notin S_j$.

This completes the proof of correctness for the case of the subset query, since we proved that for every subset query q and for every correct answer and proof output by query(), verify() always accepts.

Set difference. Let our query q be the indices i and j that refer to the difference of sets S_i and S_j. Algorithm query() outputs the proof $\Pi(q)$ and the *correct* answer $\mathsf{D} = S_i - S_j = \{y_1, y_2, \ldots, y_\delta\}$. The proof $\Pi(q)$ for a difference query contains the following parts:

1. The coefficients $b_\delta, b_{\delta-1}, \ldots, b_0$ (that relate to the difference $\{y_1, y_2, \ldots, y_\delta\}$) and the accumulation values $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$. These are always verified as in the case of intersection. See Items 1 and 2 above;
2. The completeness witnesses F_D and F_j), for proving an empty intersection of D and S_j. It is $e(\mathsf{acc}(\mathsf{D}), \mathsf{F}_\mathsf{D})e(\mathsf{acc}(S_j), \mathsf{F}_j) = e(g, g)$, by the same argument as in Item 2 in the subset query;
3. The proof for the intersection $\mathsf{I} = S_i \cap S_j$. We are using the proof as in the intersection case;
4. The value $\mathsf{L} = g^{\prod_{x\in\mathsf{I}}(x+s)} = \mathsf{acc}(\mathsf{I})$. Since $(S_i \cap S_j) \cup (S_i - S_j) = S_i$, it follows that $e(\mathsf{L}, \mathsf{acc}(\mathsf{D})) = e(\mathsf{acc}(S_i), g)$.

This completes the proof of correctness for *all* the queries supported by sets collection, since we proved that for every query q (intersection/union/subset/difference) and for every correct answer and proof output by query(), verify() always accepts.

10.2 Security

Let Adv be a computationally-bounded adversary, D_0 be a sets collection data structure consisting of m sets $S_1, S_2, \ldots, S_m$, $\mathcal{ASC} = \{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$ be our authenticated data structure scheme, k be the security parameter and $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^k)$. The adversary Adv is given the public key pk, nameley the values $\{h(.), (p, \mathbb{G}, \mathcal{G}, e, g), g^s, g^{s^2}, \ldots, g^{s^q}\}$ and unlimited access to all the algorithms of $\mathcal{ASC}$, except for setup() and update() to which he only has oracle access. The adversary initially outputs the

authenticated data structure $\mathsf{auth}(D_0)$ and the digest d_0, through an oracle call to algorithm $\mathsf{setup}()$. Then the adversary picks a polynomial number of updates (e.g., insert an element x into a set S_r) and eventually outputs the data structure D_h, the authenticated data structure $\mathsf{auth}(D_h)$ and the digest d_h through oracle access to $\mathsf{update}()$. Note that since d_h, the digest of the authenticated data structure, is produced through oracle access to $\mathsf{setup}()$ and $\mathsf{update}()$, it follows that it is the correct one. We now prove the security of each operation separately.

Intersection. Let the intersection query be a set of indices $\{1, 2, \ldots, t\}$ (wlog). The adversary Adv outputs an incorrect answer $\mathsf{I} = \{e_1, e_2, \ldots, e_\delta\} \neq S_1 \cap S_2 \cap \ldots \cap S_t$ and also a proof that consists of the following elements:

1. Coefficients $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$;
2. Accumulation values $\mathsf{acc}_1, \mathsf{acc}_2, \ldots, \mathsf{acc}_t$, along with proofs $\pi_1, \pi_2, \ldots, \pi_t$;
3. Subset witnesses $\mathsf{W}_1, \mathsf{W}_2, \ldots, \mathsf{W}_t$;
4. Completeness witnesses $\mathsf{F}_1, \mathsf{F}_2, \ldots, \mathsf{F}_t$.

We define now the following events, related to the choice of the proof above made by the adversary. Our goal will be to the express the probability of the security definition (Definition 3) in Relation 2 as a function of the following events.

- $\mathcal{E}_1$: The values $\gamma = [\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0]$ and the answer $\mathbf{e} = \{e_1, e_2, \ldots, e_\delta\}$ picked by Adv are such that $\mathsf{accept} \leftarrow \mathsf{certify}(\gamma, \mathbf{e}, \mathsf{pk})$. Event $\mathcal{E}_1$ can be partitioned into two mutually exclusive events $\mathcal{E}_{1,0}$ and $\mathcal{E}_{1,1}$, i.e, $\mathcal{E}_1 = \mathcal{E}_{1,0} \cup \mathcal{E}_{1,1}$:
  - $\mathcal{E}_{1,0}$: The coefficients $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$ are not the coefficients of the polynomial $(s + e_1)(s + e_2) \ldots (s + e_\delta)$;
  - $\mathcal{E}_{1,1}$: The coefficients $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$ are the coefficients of the polynomial $(s + e_1)(s + e_2) \ldots (s + e_\delta)$.
- $\mathcal{E}_2$: The values $\mathsf{acc}_1, \mathsf{acc}_2, \ldots, \mathsf{acc}_t$ and $\pi_1, \pi_2, \ldots, \pi_t$ picked by Adv are such that they are accepted by algorithm $\mathsf{verifyTree}()$, i.e., it is $\mathsf{accept} \leftarrow \mathsf{verifyTree}(j, \mathsf{acc}_j, \pi_j, d_h, \mathsf{pk})$, for all $j = 1, \ldots, t$. Event $\mathcal{E}_2$ can be partitioned into two mutually exclusive events $\mathcal{E}_{2,0}$ and $\mathcal{E}_{2,1}$, i.e, $\mathcal{E}_2 = \mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}$:
  - $\mathcal{E}_{2,0}$: There exists $j \in \{1, 2, \ldots, t\}$ such that $\mathsf{acc}_j \neq \mathsf{acc}(S_j)$;
  - $\mathcal{E}_{2,1}$: For all $j = 1, \ldots, t$ it is $\mathsf{acc}_j = \mathsf{acc}(S_j)$.
- $\mathcal{E}_3$: The values $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$, $\mathsf{W}_1, \mathsf{W}_2, \ldots, \mathsf{W}_t$ and $\mathsf{acc}_1, \mathsf{acc}_2, \ldots, \mathsf{acc}_t$ picked by Adv satisfy

$$e\left(\prod_{i=0}^{\delta}\left(g^{s^i}\right)^{\gamma_i}, \mathsf{W}_j\right) = e\left(\mathsf{acc}_j, g\right) \text{ for } j = 1, \ldots, t\,.$$

  Event $\mathcal{E}_3$ can be partitioned into two mutually exclusive events $\mathcal{E}_{3,0}$ and $\mathcal{E}_{3,1}$, i.e, $\mathcal{E}_3 = \mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}$:
  - $\mathcal{E}_{3,0}$: There exists $j \in \{1, 2, \ldots, t\}$ such that the opposites of the roots of the polynomial $\sum_{i=0}^{\delta} \gamma_i s^i$ are not a subset of S_j;
  - $\mathcal{E}_{3,1}$: The opposites of the roots of the polynomial $\sum_{i=0}^{\delta} \gamma_i s^i$ are a subset of S_j for all $j = 1, \ldots, t$.
- $\mathcal{E}_4$: The values $\mathsf{W}_1, \mathsf{W}_2, \ldots, \mathsf{W}_t$ and $\mathsf{F}_1, \mathsf{F}_2, \ldots, \mathsf{F}_t$ picked by Adv satisfy $\prod_{j=1}^{t} e\left(\mathsf{W}_j, \mathsf{F}_j\right) = e(g, g)$;
- $\mathcal{F}$: The answer (intersection) I picked by Adv is not correct, i.e., $\mathsf{I} = \{e_1, e_2, \ldots, e_\delta\} \neq S_1 \cap S_2 \cap \ldots \cap S_t$.

Let now $\mathcal{P}$ be the probability of Definition 3 (Relation 2), i.e., it is

$$\mathcal{P} = \Pr\left[\begin{array}{ll} \{Q, \Pi(Q), \alpha(Q), i\} \leftarrow \mathsf{Adv}(1^k, \mathsf{pk}); & \mathsf{accept} \leftarrow \mathsf{verify}(Q, \alpha(Q), \Pi(Q), d_h, \mathsf{pk}); \\ & \mathsf{reject} = \mathsf{check}(Q, \alpha(Q), D_i). \end{array}\right].$$

We recall that the authenticated data structure scheme $\mathcal{ASC}$ is secure if $\mathcal{P} \leq \nu(k)$, where $\nu(k)$ is $\mathsf{neg}(k)$. We observe that for the case of the intersection query, $\mathcal{P}$ can be expressed as the probability of the intersection

of the events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{F}$. By using simple probability calculus, this can be written as

$$\begin{aligned}
\mathcal{P} &= \Pr\left[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{E}_4 \cap \mathcal{F}\right] = \Pr\left[(\mathcal{E}_{1,0} \cup \mathcal{E}_{1,1}) \cap (\mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}) \cap (\mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}) \cap \mathcal{E}_4 \cap \mathcal{F}\right] \\
&\leq \Pr[\mathcal{E}_{1,0}] + \Pr\left[\mathcal{E}_{1,1} \cap (\mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}) \cap (\mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}) \cap \mathcal{E}_4 \cap \mathcal{F}\right] \\
&\leq \Pr[\mathcal{E}_{1,0}] + \Pr[\mathcal{E}_{2,0}] + \Pr\left[\mathcal{E}_{1,1} \cap \mathcal{E}_{2,1} \cap (\mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}) \cap \mathcal{E}_4 \cap \mathcal{F}\right] \\
&\leq \Pr[\mathcal{E}_{1,0}] + \Pr[\mathcal{E}_{2,0}] + \Pr[\mathcal{E}_{3,0} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}] + \Pr\left[\mathcal{E}_{1,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{3,1} \cap \mathcal{E}_4 \cap \mathcal{F}\right] \\
&\leq \Pr[\mathcal{E}_{1,0}] + \Pr[\mathcal{E}_{2,0}] + \Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}] + \Pr\left[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}\right] .
\end{aligned}$$

We compute each such probability separately:

1. $\Pr[\mathcal{E}_{1,0}]$ is $\mathsf{neg}(k)$ by Lemma 3;
2. $\Pr[\mathcal{E}_{2,0}]$ is $\mathsf{neg}(k)$ by Lemma 5;
3. $\Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}]$: For this event we note that the event $\mathcal{E}_{3,0}$ is conditioned on the event $\mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}$. This condition allows us to replace acc_j with $\mathsf{acc}(S_j)$ (due to $\mathcal{E}_{2,1}$) and $\sum_{i=0}^{\delta} \gamma_i s^i$ with $\prod_{x \in \mathsf{I}}(x+s)$ (due to $\mathcal{E}_{1,1}$) in the event $\mathcal{E}_{3,0}$. Therefore the event $\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}$ is the event

$$e\left(g^{\prod_{x \in \mathsf{I}}(x+s)}, \mathsf{W}_j\right) = e\left(\mathsf{acc}(S_j), g\right) \wedge \mathsf{I} \nsubseteq S_j \text{ for some } j \in \{1, 2, \ldots, t\} .$$

   This event implies breaking the bilinear q-strong Diffie-Hellman assumption (Assumption 1), by Lemma 1. Therefore the probability $\Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}]$ is $\mathsf{neg}(k)$;
4. $\Pr[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}]$: For this event we note that the event $\mathcal{E}_4$ is conditioned on the event $\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}$. This condition allows us to replace acc_j with $\mathsf{acc}(S_j)$ (due to $\mathcal{E}_{2,1}$) and $\sum_{i=0}^{\delta} \gamma_i s^i$ with $\prod_{x \in \mathsf{I}}(x+s)$ (due to $\mathcal{E}_{1,1}$) in the event $\mathcal{E}_{3,1}$. Therefore, the event $\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}$ is the event

$$e\left(g^{\prod_{x \in \mathsf{I}}(x+s)}, \mathsf{W}_j\right) = e\left(\mathsf{acc}(S_j), g\right) \wedge \mathsf{I} \subseteq S_j \text{ for all } j = 1, 2, \ldots, t .$$

   This is equivalent to writing W_j as the subset witness $\mathsf{W}_{\mathsf{I},S_j}$, i.e.,

$$\mathsf{W}_j = g^{\prod_{x \in S_j - \mathsf{I}}(x+s)} = g^{P_j(s)} . \tag{12}$$

Note now that $\mathcal{E}_4$ is also conditioned on $\mathcal{F}$. Therefore I is has to be incorrect. Specifically, since $\mathsf{I} \subseteq S_j$ for all $j = 1, \ldots, t$, it follows that I does not contain all the elements of the intersection, i.e., it is *incomplete*. Thus the polynomials $P_1(s), P_2(s), \ldots, P_t(s)$ (Relation 12) have at least one common factor, say $(s+r)$ and it holds $P_j(s) = (s+r)Q_j(s)$ for some polynomials $Q_j(s)$—computable in polynomial time—, for all $j = 1, \ldots, t$. Therefore the event $\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}$ implies that

$$\begin{aligned}
e(g,g) &= \prod_{j=1}^{t} e\left(\mathsf{W}_j, \mathsf{F}_j\right) = \prod_{j=1}^{t} e\left(g^{P_j(s)}, \mathsf{F}_j\right) = \prod_{j=1}^{t} e\left(g^{(s+r)Q_j(s)}, \mathsf{F}_j\right) \\
&= \prod_{j=1}^{t} e\left(g^{Q_j(s)}, \mathsf{F}_j\right)^{(s+r)} = \left(\prod_{j=1}^{t} e\left(g^{Q_j(s)}, \mathsf{F}_j\right)\right)^{(s+r)} .
\end{aligned}$$

Therefore we can derive an $1/(s+r)$-th of $e(g,g)$ as

$$e(g,g)^{\frac{1}{s+r}} = \prod_{j=1}^{t} e\left(g^{Q_j(s)}, \mathsf{F}_j\right) .$$

This implies breaking the bilinear q-strong Diffie-Hellman assumption for the setting $(p, \mathbb{G}, \mathcal{G}, e, g)$ (Assumption 1). By Assumption 1, this probability is $\mathsf{neg}(k)$, and therefore $\Pr[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}]$ is $\mathsf{neg}(k)$. Thus the total probability $\mathcal{P}$ is $\mathsf{neg}(k)$. This concludes the proof for the security of an intersection query.

Union. Let the union query be a set of indices $\{1, 2, \ldots, t\}$ (wlog). The adversary Adv outputs an incorrect answer $\mathsf{U} = \{e_1, e_2, \ldots, e_\delta\} \neq S_1 \cup S_2 \cup \ldots \cup S_t$ and also a proof that consists of the following elements:

1. Coefficients $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$;
2. Accumulation values $\mathsf{acc}_1, \mathsf{acc}_2, \ldots, \mathsf{acc}_t$, along with proofs $\pi_1, \pi_2, \ldots, \pi_t$;
3. For each element $e_i \in \mathsf{U}$, membership witnesses $\mathsf{W}_{i,j}$ with reference to some set S_j, where $1 \leq j \leq t$;
4. Subset witnesses $\mathsf{W}_1, \mathsf{W}_2, \ldots, \mathsf{W}_t$ that prove that U is a subset of S_j, for all $j = 1, 2 \ldots, t$.

We define now the following events, related to the choice of the proof above made by the adversary. Our goal will be to the express the probability of the security definition (Definition 3) in Relation 2 as a function of the following events.

- $\mathcal{E}_1, \mathcal{E}_{1,0}, \mathcal{E}_{1,0}$: Same as in intersection;
- $\mathcal{E}_2, \mathcal{E}_{2,0}, \mathcal{E}_{2,0}$: Same as in intersection;
- $\mathcal{E}_3$: The values $\{e_1, e_2, \ldots, e_\delta\}, \mathsf{W}_{1,j_1}, \mathsf{W}_{1,j_2}, \ldots, \mathsf{W}_{1,j_\delta}$ picked by Adv satisfy

$$e\left(\mathsf{W}_{i,j_i}, g^s g^{e_i}\right) = e(\mathsf{acc}_{j_i}, g) \text{ for all } i = 1, \ldots, \delta \text{ and } j_i \in \{1, 2, \ldots, t\}\,.$$

  Event $\mathcal{E}_3$ can be partitioned into two mutually exclusive events $\mathcal{E}_{3,0}$ and $\mathcal{E}_{3,1}$, i.e, $\mathcal{E}_3 = \mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}$:
  - $\mathcal{E}_{3,0}$: There exists $i \in \{1, 2, \ldots, \delta\}$ such that $e_i \notin S_{j_i}$;
  - $\mathcal{E}_{3,1}$: For all $i = 1, 2, \ldots, \delta$ it is $e_i \in S_{j_i}$.
- $\mathcal{E}_4$: The values $\mathsf{W}_1, \mathsf{W}_2, \ldots, \mathsf{W}_t$, $\mathsf{acc}_1, \mathsf{acc}_2, \ldots, \mathsf{acc}_t$ and $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$ picked by Adv satisfy

$$e\left(\mathsf{W}_j, \mathsf{acc}_j\right) = e\left(\prod_{i=0}^{\delta} \left(g^{s^i}\right)^{\gamma_i}, g\right)\,.$$

- $\mathcal{F}$: The answer (union) U picked by Adv is not correct, i.e., $\mathsf{U} = \{e_1, e_2, \ldots, e_\delta\} \neq S_1 \cup S_2 \cup \ldots \cup S_t$.

Similarly with the intersection security proof, let $\mathcal{P}$ be the probability of Definition 3 (Relation 2). We observe that for the case of the union query, $\mathcal{P}$ can be expressed as the probability of the intersection of the events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{F}$. By using simple probability calculus (and similarly with the intersection security proof), this can be written as

$$\begin{aligned}\mathcal{P} &= \Pr\left[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{E}_4 \cap \mathcal{F}\right] = \Pr\left[(\mathcal{E}_{1,0} \cup \mathcal{E}_{1,1}) \cap (\mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}) \cap (\mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}) \cap \mathcal{E}_4 \cap \mathcal{F}\right] \\ &\leq \Pr[\mathcal{E}_{1,0}] + \Pr[\mathcal{E}_{2,0}] + \Pr[\mathcal{E}_{3,0}|\mathcal{E}_{2,1}] + \Pr\left[\mathcal{E}_4|\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}\right]\,.\end{aligned}$$

We compute each such probability separately:

1. $\Pr[\mathcal{E}_{1,0}]$ is $\mathsf{neg}(k)$ by Lemma 3;
2. $\Pr[\mathcal{E}_{2,0}]$ is $\mathsf{neg}(k)$ by Lemma 5;
3. $\Pr[\mathcal{E}_{3,0}|\mathcal{E}_{2,1}]$: For this event we note that the event $\mathcal{E}_{3,0}$ is conditioned on the event $\mathcal{E}_{2,1}$. This condition allows us to replace acc_j with $\mathsf{acc}(S_j)$ in the event $\mathcal{E}_{3,0}$. Therefore the event $\mathcal{E}_{3,0}|\mathcal{E}_{2,1}$ is the event

$$e\left(\mathsf{W}_{i,j_i}, g^s g^{e_i}\right) = e(\mathsf{acc}(S_{j_i}), g) \wedge \exists i \in \{1, 2, \ldots, \delta\} \wedge j_i \in \{1, 2, \ldots, t\} : e_i \notin S_{j_i}\,.$$

   This event implies breaking the bilinear q-strong Diffie-Hellman assumption (Assumption 1), by Lemma 1. Therefore the probability $\Pr[\mathcal{E}_{3,0}|\mathcal{E}_{2,1}]$ is $\mathsf{neg}(k)$;
4. $\Pr\left[\mathcal{E}_4|\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}\right]$: For this event we note that the event $\mathcal{E}_4$ is conditioned on the event $\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}$. This condition allows us to replace acc_j with $\mathsf{acc}(S_j)$ (due to $\mathcal{E}_{2,1}$) and $\sum_{i=0}^{\delta} \gamma_i s^i$ with $\prod_{x \in \mathsf{U}}(x + s)$ (due to $\mathcal{E}_{1,1}$) in the event $\mathcal{E}_4$. Therefore, the event $\mathcal{E}_4|\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}$ is the event

$$e\left(\mathsf{W}_j, \mathsf{acc}(S_j)\right) = e\left(g^{\prod_{x \in \mathsf{U}}(x+s)}, g\right)\,. \tag{13}$$

Note now that $\mathcal{E}_4$ is also conditioned on $\mathcal{E}_{3,1}$. Thus it holds that all elements $e_i \in \mathsf{U}$ belong to some S_{j_i}. Therefore the reported union cannot contain extra elements. Also, $\mathcal{E}_4$ is conditioned on $\mathcal{F}$ (incorrect union). Therefore the reported union must contain less elements and there should be an S_j $(1 \leq j \leq t)$ that contains an r such that $r \notin \mathsf{U}$. Therefore since Relation 13 holds, we can find $P(s)$, $Q(s)$ and α such that

$$e\left(\mathsf{W}_j, \mathsf{acc}(S_j)\right) = e(\mathsf{W}_j, g)^{(s+r)P(s)} = e\left(g^{\prod_{x \in \mathsf{U}}(x+s)}, g\right) = e(g,g)^{(s+r)Q(s)+\alpha} .$$

Therefore we can derive an $1/(s+r)$-th of $e(g,g)$ as

$$e(g,g)^{\frac{1}{s+r}} = e(g, \mathsf{W}_{S_j})^{P(s)/\alpha} e(g,g)^{-Q(s)/\alpha} .$$

This implies breaking the bilinear q-strong Diffie-Hellman assumption for the setting $(p, \mathbb{G}, \mathcal{G}, e, g)$ (Assumption 1). By Assumption 1, this probability is $\mathsf{neg}(k)$, and therefore $\Pr[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1} \cap \mathcal{F}]$ is $\mathsf{neg}(k)$. Thus the total probability $\mathcal{P}$ is $\mathsf{neg}(k)$. This concludes the proof for the security of a union query.

Subset. Let the subset query be "is $S_i \subseteq S_j$?". For a positive answer, the adversary Adv outputs an incorrect answer "false" and also a proof that consists of the following elements:

1. Accumulation values acc_i and acc_j along with proofs π_i and π_j;
2. A membership witness $\mathsf{W}_{i,j}$ with reference to set S_j.

We define now the following events, related to the choice of the proof above made by the adversary. Our goal will be to the express the probability of the security definition (Definition 3) in Relation 2 as a function of the following events.

- $\mathcal{E}_2$: The values acc_i, acc_j, π_i and π_j picked by Adv are such that they are accepted by algorithm $\mathsf{verifyTree}()$, i.e., $\mathsf{accept} \leftarrow \mathsf{verifyTree}(i, \mathsf{acc}_i, \pi_i, d_h, \mathsf{pk})$ and $\mathsf{accept} \leftarrow \mathsf{verifyTree}(j, \mathsf{acc}_j, \pi_j, d_h, \mathsf{pk})$. Event $\mathcal{E}_2$ can be partitioned into two mutually exclusive events $\mathcal{E}_{2,0}$ and $\mathcal{E}_{2,1}$, i.e, $\mathcal{E}_2 = \mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}$:
  - $\mathcal{E}_{2,0}$: $\mathsf{acc}_i \neq \mathsf{acc}(S_j)$ or $\mathsf{acc}_j \neq \mathsf{acc}(S_i)$;
  - $\mathcal{E}_{2,1}$: $\mathsf{acc}_i = \mathsf{acc}(S_i)$ and $\mathsf{acc}_j = \mathsf{acc}(S_j)$.
- $\mathcal{E}_3$: The values acc_i, acc_j and $\mathsf{W}_{i,j}$ picked by Adv satisfy $e\left(\mathsf{W}_{i,j}, \mathsf{acc}_i\right) = e\left(\mathsf{acc}_j, g\right)$.
- $\mathcal{F}$: $S_i \nsubseteq S_j$.

Similarly with the intersection security proof, let $\mathcal{P}$ be the probability of Definition 3 (Relation 2). We observe that for the case of the positive subset query, $\mathcal{P}$ can be expressed as the probability of the intersection of the events $\mathcal{E}_2, \mathcal{E}_3, \mathcal{F}$. By using simple probability calculus (and similarly with the intersection security proof), this can be written as

$$\mathcal{P} = \Pr\left[\mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{F}\right] = \Pr\left[(\mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}) \cap \mathcal{E}_3 \cap \mathcal{F}\right] \leq \Pr[\mathcal{E}_{2,0}] + \Pr\left[\mathcal{E}_3 | \mathcal{E}_{2,1} \cap \mathcal{F}\right] .$$

We compute each such probability separately:

1. $\Pr[\mathcal{E}_{2,0}]$ is $\mathsf{neg}(k)$ by Lemma 5;
2. $\Pr[\mathcal{E}_3 | \mathcal{E}_{2,1} \cap \mathcal{F}]$: For this event we note that the event $\mathcal{E}_3$ is conditioned on the event $\mathcal{E}_{2,1} \cap \mathcal{F}$. This condition allows us to replace acc_i with $\mathsf{acc}(S_i)$ and acc_j with $\mathsf{acc}(S_j)$ in the event $\mathcal{E}_3$. Therefore the event $\mathcal{E}_3 | \mathcal{E}_{2,1} \cap \mathcal{F}$ is the event

$$e\left(\mathsf{W}_{i,j}, \mathsf{acc}(S_i)\right) = e\left(\mathsf{acc}(S_j), g\right) \wedge S_i \nsubseteq S_j .$$

This event implies breaking the bilinear q-strong Diffie-Hellman assumption (Assumption 1), by Lemma 1. Therefore the probability $\Pr[\mathcal{E}_3 | \mathcal{E}_{2,1} \cap \mathcal{F}]$ is $\mathsf{neg}(k)$;

This concludes the security proof for the case of the positive subset query. For a negative answer, the adversary Adv outputs an incorrect answer "true" and also a proof that consists of the following elements:

1. Accumulation values acc_i and acc_j along with proofs π_i and π_j;
2. An element y;
3. A membership witness W_y for element y;
4. Completeness witnesses F_y and F_j for element y.

We define now the following events, related to the choice of the proof above made by the adversary. Our goal will be to the express the probability of the security definition (Definition 3) in Relation 2 as a function of the following events.

- $\mathcal{E}_2$: Same as in the positive answer;
- $\mathcal{E}_3$: The values acc_i, W_y and y picked by Adv are such that $e(\mathsf{W}_y, g^s g^y) = e(\mathsf{acc}_i, g)$. Event $\mathcal{E}_3$ can be partitioned into two mutually exclusive events $\mathcal{E}_{3,0}$ and $\mathcal{E}_{3,1}$, i.e, $\mathcal{E}_3 = \mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}$:
  - $y \in S_i$;
  - $y \notin S_i$.
- $\mathcal{E}_4$: The values y, F_y, acc_j and F_j picked by Adv are such that $e(g^y g^s, \mathsf{F}_y) e(\mathsf{acc}_j, \mathsf{F}_j) = e(g, g)$;
- $\mathcal{F}$: $S_i \subseteq S_j$.

Similarly with the intersection security proof, let $\mathcal{P}$ be the probability of Definition 3 (Relation 2). We observe that for the case of the negative subset query, $\mathcal{P}$ can be expressed as the probability of the intersection of the events $\mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{F}$. By using simple probability calculus (and similarly with the intersection security proof), this can be written as

$$\begin{aligned} \mathcal{P} &= \Pr\left[\mathcal{E}_4 \cap \mathcal{E}_3 \cap \mathcal{E}_2 \cap \mathcal{F}\right] = \Pr\left[\mathcal{E}_4 \cap (\mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}) \cap (\mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}) \cap \mathcal{F}\right] \\ &\leq \Pr[\mathcal{E}_{2,0}] + \Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1}] + \Pr\left[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}\right] . \end{aligned}$$

We compute each such probability separately:

1. $\Pr[\mathcal{E}_{2,0}]$ is $\mathsf{neg}(k)$ by Lemma 5;
2. $\Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1}]$: For this event we note that the event $\mathcal{E}_{3,0}$ is conditioned on the event $\mathcal{E}_{2,1}$. This condition allows us to replace acc_i with $\mathsf{acc}(S_i)$ in the event $\mathcal{E}_{3,0}$. Therefore the event $\mathcal{E}_{3,0} | \mathcal{E}_{2,1}$ is the event

   $$e\left(\mathsf{W}_y, g^s g^y\right) = e\left(\mathsf{acc}(S_i), g\right) \wedge y \notin S_i \, .$$

   This event implies breaking the bilinear q-strong Diffie-Hellman assumption (Assumption 1), by Lemma 1. Therefore the probability $\Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1}]$ is $\mathsf{neg}(k)$;
3. $\Pr\left[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}\right]$. Due to the condition on $\mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}$ this is the event

   $$e\left(g^y g^s, \mathsf{F}_y\right) e\left(\mathsf{acc}(S_j), \mathsf{F}_j\right) = e(g, g) \wedge S_i \subseteq S_j \, .$$

   Since we have the condition on $\mathcal{E}_{3,1} \cap \mathcal{F}$ ($y \in S_i$ and $S_i \subseteq S_j$), it must be that $y \in S_j$. Therefore there exists a polynomial $Q(s)$ such that

   $$e\left(g^y g^s, \mathsf{F}_y\right) e\left(g^{Q(s)(y+s)}, \mathsf{F}_j\right) = e(g, g) \, .$$

   Therefore we can derive an $1/(s+r)$-th of $e(g, g)$ as

   $$e(g, g)^{\frac{1}{s+r}} = e\left(g, \mathsf{F}_y\right) e\left(g^{Q(s)}, \mathsf{F}_j\right) \, .$$

   This implies breaking the bilinear q-strong Diffie-Hellman assumption for the setting $(p, \mathbb{G}, \mathcal{G}, e, g)$ (Assumption 1). By Assumption 1, this probability is $\mathsf{neg}(k)$, and therefore $\Pr\left[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}\right]$ is $\mathsf{neg}(k)$. Thus the total probability $\mathcal{P}$ is $\mathsf{neg}(k)$. This concludes the proof for the security of a negative subset query.

Set difference. Let the difference query be $\mathsf{D} = S_i - S_j$. The adversary Adv outputs an incorrect answer $\mathsf{D} = \{e_1, e_2, \ldots, e_\delta\} \neq S_i - S_j$ and also a proof that consists of the following elements:

1. Coefficients $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$;
2. Accumulation values acc_i and acc_j along with proofs π_i and π_j;
3. A subset witness $\mathsf{W}_{\mathsf{D},S_i}$;
4. A proof $(\mathsf{W}_i, \mathsf{W}_j, \mathsf{F}_i, \mathsf{F}_j)$ for the intersection $S_i \cap S_j$.

We define now the following events, related to the choice of the proof above made by the adversary. Our goal will be to the express the probability of the security definition (Definition 3) in Relation 2 as a function of the following events.

- $\mathcal{E}_1$: Same as in intersection;
- $\mathcal{E}_2$: Same as in subset;
- $\mathcal{E}_3$: The values $\gamma_\delta, \gamma_{\delta-1}, \ldots, \gamma_0$, $\mathsf{W}_{\mathsf{D},S_i}$ and acc_i picked by Adv satisfy

$$e\left(\mathsf{W}_{\mathsf{D},S_i}, \prod_{i=0}^{\delta}\left(g^{s^i}\right)^{\gamma_i}\right) = e(\mathsf{acc}_i, g)\,.$$

  Event $\mathcal{E}_3$ can be partitioned into two mutually exclusive events $\mathcal{E}_{4,0}$ and $\mathcal{E}_{4,1}$, i.e, $\mathcal{E}_4 = \mathcal{E}_{4,0} \cup \mathcal{E}_{4,1}$:
  - $\mathcal{E}_{3,0}$: $\mathsf{D} \nsubseteq S_i$;
  - $\mathcal{E}_{3,1}$: $\mathsf{D} \subseteq S_i$;
- $\mathcal{E}_4$: The values $\mathsf{W}_{\mathsf{D},S_i}, \mathsf{acc}_i, \mathsf{acc}_j, \mathsf{W}_i, \mathsf{W}_j, \mathsf{F}_i$ and F_j picked by Adv are such that the respective tests for the intersection of S_i and S_j are satisfied, i.e.,

$$e(\mathsf{W}_i, \mathsf{W}_{\mathsf{D},S_i}) = e(\mathsf{acc}_i, g) \wedge e(\mathsf{W}_j, \mathsf{W}_{\mathsf{D},S_i}) = e(\mathsf{acc}_j, g) \wedge e(\mathsf{W}_i, \mathsf{F}_i)e(\mathsf{W}_j, \mathsf{F}_j) = e(g, g)\,.$$

- $\mathcal{F}$: The difference D is incorrect, i.e., $\mathsf{D} \neq S_i - S_j$.

Similarly with the intersection security proof, let $\mathcal{P}$ be the probability of Definition 3 (Relation 2). We observe that for the case of the difference query, $\mathcal{P}$ can be expressed as the probability of the intersection of the events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{F}$. By using simple probability calculus (and similarly with the intersection security proof), this can be written as

$$\begin{aligned}
\mathcal{P} &= \Pr\left[\mathcal{E}_4 \cap \mathcal{E}_3 \cap \mathcal{E}_2 \cap \mathcal{E}_1 \cap \mathcal{F}\right] \\
&= \Pr\left[\mathcal{E}_4 \cap (\mathcal{E}_{3,0} \cup \mathcal{E}_{3,1}) \cap (\mathcal{E}_{2,0} \cup \mathcal{E}_{2,1}) \cap (\mathcal{E}_{1,0} \cup \mathcal{E}_{1,1}) \cap \mathcal{F}\right] \\
&\leq \Pr[\mathcal{E}_{1,0}] + \Pr[\mathcal{E}_{2,0}] + \Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}] + \Pr[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}]\,.
\end{aligned}$$

We compute each such probability separately:

1. $\Pr[\mathcal{E}_{1,0}]$ is $\mathsf{neg}(k)$ by Lemma 3;
2. $\Pr[\mathcal{E}_{2,0}]$ is $\mathsf{neg}(k)$ by Lemma 5;
3. $\Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}]$. For the event $\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}$, by replacing the values of the conditions, we get

$$e\left(\mathsf{W}_{\mathsf{D},S_i}, g^{\prod_{x \in \mathsf{D}}(x+s)}\right) = e(\mathsf{acc}(S_i), g) \wedge \mathsf{D} \nsubseteq S_i\,.$$

   This event implies breaking the bilinear q-strong Diffie-Hellman assumption (Assumption 1), by Lemma 1. Therefore the probability $\Pr[\mathcal{E}_{3,0} | \mathcal{E}_{2,1} \cap \mathcal{E}_{1,1}]$ is $\mathsf{neg}(k)$;
4. $\Pr[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}]$. By the conditions, since $\mathsf{D} \subseteq S_i$ we can write $\mathsf{W}_{\mathsf{D},S_i} = g^{\prod_{x \in S_i - \mathsf{D}}(x+s)}$. Therefore the event is equivalent to

$$\begin{aligned}
e\left(\mathsf{W}_i, g^{\prod_{x \in S_i - \mathsf{D}}(x+s)}\right) = e\left(\mathsf{acc}(S_i), g\right) \quad &\wedge \quad e\left(\mathsf{W}_j, g^{\prod_{x \in S_i - \mathsf{D}}(x+s)}\right) = e\left(\mathsf{acc}(S_j), g\right) \\
&\wedge \quad e\left(\mathsf{W}_i, \mathsf{F}_i\right) e\left(\mathsf{W}_j, \mathsf{F}_j\right) = e(g, g)\,.
\end{aligned}$$

We have already proved (intersection proof) that the probability that the above event holds and $S_i - \mathsf{D} \neq S_i \cap S_j$ is $\mathsf{neg}(k)$. However, the event $S_i - \mathsf{D} \neq S_i \cap S_j$ is equivalent with the event $\mathsf{D} \neq S_i - S_j$, which is our event $\mathcal{F}$. Therefore the probability $\Pr[\mathcal{E}_4 | \mathcal{E}_{3,1} \cap \mathcal{E}_{2,1} \cap \mathcal{F}]$ is $\mathsf{neg}(k)$.

This completes the proof of security for all the queries of the sets collection data structure.

Complexity. The complexity proofs are given in the *description of the algorithms*.

11 Applications

Timestamped keyword-search. As we saw, in timestamped keyword-search, the embedded timestamp technique cannot work since the intersection can be a lot bigger than the set resulted after the application of the local filtering, making this straightforward solution inefficient. We now describe an algorithmic construction to solve this problem. Let $t_1, t_2, \ldots, t_r$ be the discrete timestamps that we are interested in (t_i can be viewed as a certain day of the month). We define a new sets collection data structure as follows: Imagine $t_1, t_2, \ldots, t_r$ are the leaves of a binary tree. We build a *segment tree* [38] on top of these timestamps as follows: Each leaf storing timestamp t_i contains the documents (e.g., email messages) that were received at time t_i. Moreover, the internal nodes of the binary tree contain the documents that correspond to the union (note that this union does not have any common elements) of the documents contained in the children's nodes, recursively defining in this way sets of documents for all the nodes of the tree. Therefore we end up with a new sets collection data structure that is built on top of these $2r - 1$ sets (one set per internal tree node of the tree), namely the sets $T_1, T_2, \ldots, T_{2r-1}$. The timestamped keyword-search is therefore verified by two sets collection data structures, one built on the text terms, namely the sets $S_1, S_2, \ldots, S_m$, and one built on top of the sets of the timestamps, namely the sets $T_1, T_2, \ldots, T_{2r-1}$. Define now the extension of two timestamps $\mathsf{ext}(t_1, t_2)$ to be the *set of sets* T_i that "cover" the interval $[t_1, t_2]$, i.e., namely the set that contains sets the union of which equals the set of all timestamps in $[t_1, t_2]$. One can easily see that for every $1 \leq t_1 \leq t_2 \leq r$, it is $|\mathsf{ext}(t_1, t_2)| = O(\log r)$.

Suppose now we want to verify the documents that contain terms m_1 and m_2 and which were received between t_1 and t_2. All we have to do is to verify the intersection of the following sets: (a) the union of sets in $\mathsf{ext}(t_1, t_2)$, (b) S_1 (set that refers to term m_1) and, (c) S_2 (set that refers to term m_2). Let $T_1, T_2, \ldots, T_\ell$ be the disjoint sets that are contained in $\mathsf{ext}(t_1, t_2)$, where $\ell = O(\log r)$. The answer to the query is the set $(S_1 \cap S_2) \cap (T_1 \cup T_2 \cup \ldots \cup T_\ell)$ which can be written as $(S_1 \cap S_2 \cap T_1) \cup (S_1 \cap S_2 \cap T_2) \cup \ldots \cup (S_1 \cap S_2 \cap T_\ell)$. Since T_i are disjoint, each term of the union contributes at least one new term to the answer, and therefore we can verify this query in a nearly *operation-sensitive* way by authenticating $\log r$ intersections separately (note there is an extra $O(\log r)$ multiplicative factor in the complexities of Theorem 2). Therefore:

Theorem 2 *Consider a collection of m sets $S_1, \ldots, S_m$ and let $M = \sum_{i=1}^{m} |S_i|$ and $0 < \epsilon < 1$. For a query operation involving in a time interval $[t_1, t_2]$, let t be the number of involved sets, N be the sum of the sizes of the involved sets, and δ be the answer size. There exists an authenticated data structure scheme $\mathcal{TKS} = \{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$ for a timestamped keyword-search data structure D with the following properties: (1) it is correct and secure according to Definitions 2 and 3 and based on the bilinear q-strong Diffie-Hellman assumption; (2) The access complexity of algorithm (i) $\mathsf{genkey}()$ is $O(1)$; (ii) $\mathsf{setup}()$ is $O(m + r + M)$; (iii) $\mathsf{update}()$ is $O(1)$ outputting information upd of $O(1)$ group complexity; (iv) $\mathsf{refresh}()$ is $O(1)$; (3) For a time-stamped keyword-search query, algorithm $\mathsf{query}()$ has $O(N \log r \log^2 N \log\log N + t(m+r)^\epsilon \log(m+r))$ access complexity, algorithm $\mathsf{verify}()$ has $O(t \log r + \delta)$ access complexity and the proof $\Pi(q)$ has $O(t \log r + \delta)$ group complexity; (4) The group complexity of the authenticated data structure $\mathsf{auth}(D)$ is $O(m + r + M)$.*

Note that in the above theorem we do not have a result concerning the verification of union with timestamps. This is due to the following: Using the same notation as we did for the intersection, the answer to the union query, would be the set $(S_1 \cup S_2) \cap (T_1 \cup T_2 \cup \ldots \cup T_\ell)$. The nature of the answer does not allow for any further algebraic processing and therefore in order to authenticate the whole expression, one

needs to verify the two unions separately. This leads to a solution that is not operation-sensitive, therefore the operation-sensitive verification of this type of queries cannot be achieved with our method—at least in a way similar to the techniques we have used so far. The same applies for the difference queries.

Equi-join queries. Finally, we show how our technique can be used in efficient verification of *database queries*, such as *equi-join*. Let $R_1(\alpha, r_{11}, \ldots, r_{1w}), \ldots, R_m(\alpha, r_{m1}, \ldots, r_{mw})$ be m relational tables, that have up to n tuples each, and which share a common attribute α. We now want to compute the equi-join query on the common attribute α on any subset of t of them. This is basically an intersection that can be authenticated by building our scheme on top of the attributes α, for all relations $R_1, R_2, \ldots, R_m$.

To handle duplicate values for the attribute α, we build our authentication scheme on top of distinct α values for all the relations R_i, $i = 1, \ldots, m$ and we keep a separate structure that maps α to all the related records, for all relations R_i, $i = 1, \ldots, m$. This authenticated structure can be a bilinear-map accumulated value that adds (i.e., the respective witness) only constant overhead per relation (1024 bits in practice) to the proof size, for each element that appears in the equi-join answer. We note that the verification of these type of queries have also been studied in [41]. Therefore we have the following theorem:

Theorem 3 *Consider a collection of m relational tables R_i for $i = 1, \ldots, m$ and let $M = \sum_{i=1}^{m} |R_i|$. For an equi-join query, let t be the number of involved relational tables, N be the sum of the sizes of the involved relational tables, and δ be the answer size. There exists an authenticated data structure scheme $\mathcal{EQJ} = \{\mathsf{genkey}, \mathsf{setup}, \mathsf{update}, \mathsf{refresh}, \mathsf{query}, \mathsf{verify}\}$ for a data structure answering equi-join queries D with the following properties: (1) it is correct and secure according to Definitions 2 and 3 and based on the bilinear q-strong Diffie-Hellman assumption; (2) The access complexity of algorithm (i) $\mathsf{genkey}()$ is $O(1)$; (ii) $\mathsf{setup}()$ is $O(m+M)$; (iii) $\mathsf{update}()$ is $O(1)$ outputting information upd of $O(1)$ group complexity; (iv) $\mathsf{refresh}()$ is $O(1)$; (3) For an equi-join query, algorithm $\mathsf{query}()$ has $O(N \log^2 N \log\log N + tm^{\epsilon} \log m)$ access complexity, algorithm $\mathsf{verify}()$ has $O(t+\delta)$ access complexity and the proof $\Pi(q)$ has $O(t+\delta)$ group complexity; (4) The group complexity of the authenticated data structure $\mathsf{auth}(D)$ is $O(m+M)$.*

12 Analysis

In this section we analyze the costs needed by our solution and compare with experimental results from other works. For bilinear maps and generic-group operations in the bilinear-map accumulator, we used the PBC library [1], a library for pairing-based cryptography, interfaced with C.

12.1 System setup

We choose our system parameters as follows. First of all, type A pairings are used, as described in [24]. These pairings are constructed on the curve $y^2 = x^3 + x$ over the base field $\mathbb{F}_q$, where q is a prime number. The multiplicative cyclic group $\mathbb{G}$ we are using is a subgroup of points in $E(\mathbb{F}_q)$, namely a subset of those points of $\mathbb{F}_q$ that belong to the elliptic curve E. Therefore this pairing is symmetric. The order of $E(\mathbb{F}_q)$ is $q+1$ and the order of the group $\mathbb{G}$ is some prime factor p of $q+1$. The group of the output of the bilinear map $\mathcal{G}$ is a subgroup of $\mathbb{F}_{q^2}$.

In order to instantiate type A pairings in the PBC library, we have to choose the size of the primes q and p. The main constraint in choosing the bit-sizes of q and p is that we want to make sure that discrete logarithm is difficult in $\mathbb{G}$ (that has order p) and in $\mathbb{F}_{q^2}$. Typical values are 160 bits for p and 512 bits for q. We use the typical value for the size of q, i.e., 512 bits. Note that with this choice of parameters the size of the elements in $\mathbb{G}$ (which have the form (x, y), i.e., points on the elliptic curve) is 1024 bits. Finally, let's assume that the accumulation tree that is built on top of the set digests, has two levels, i.e., $\epsilon = 0.5$.

12.2 Communication cost

Here we analyze the communication cost that our scheme has for an intersection of *two* sets. Let's assume that the size of the reported intersection is δ. Then as we saw in Section 3, the proof (apart from the answer itself), consists of the following values: (a) Two subset witnesses, two completeness witnesses, two

accumulation values (each one of the accumulation values comes with two proof elements of two group elements each that serve as a proof for it). Therefore the size of all these elements, which are all elements of group $\mathbb{G}$, is not dependent on the size of the intersection and is equal to $2 \times (1024 + 1024 + 1024 + 4 \times 1024)/8 = 1792$ bytes; (b) The coefficients $b_i \in \mathbb{Z}_p$ (we recall p is 160 bits long) of the intersection, for $i = 1, \ldots, \delta$. These have size $160\delta/8 = 20\delta$ bytes. Therefore the total communication cost is a linear function of δ, i.e., the function $1792 + 20\delta$ (in bytes).

We now compare the communication cost of our scheme with the analysis made in [29]. In Table 2 we compare with the results presented in Table IV of [29] where various set sizes n_1 and n_2 are used and the size of the intersection δ is always $0.01n_2$. Note that in most cases, our communication cost is a lot less than the one reported in [29]. More importantly, it is *not* dependent on the size of the sets participating in the intersection. In cases that our cost is worse, it is due to the big constants enforced by the use of bilinear pairings and accumulators.

Table 2: Comparison of a 2-intersection communication overhead (proof size) of the scheme presented in [29] with our scheme. Here n_1 and n_2 are the sets sizes that are intersected and δ is the size of the intersection.

n_1	n_2	δ	KB [29]	KB (this work)
1000	1000	10	3.34	1.94
1000	100	1	1.68	1.76
1000	10	0	1.01	1.75
1000	1	0	0.46	1.75
10000	10000	100	26.88	3.70
10000	1000	10	12.15	1.94
10000	100	1	6.86	1.76
10000	10	0	3.08	1.75
100000	100000	1000	263.25	21.28
100000	10000	100	116.13	3.70
100000	1000	10	63.18	1.94
100000	100	1	26.69	1.76

Table 3: Comparison of an equi-join communication overhead (proof size) of the scheme presented in [29] with our scheme. Tuple size is in bytes.

tuple size	32	64	128	256	512
MB [41]	15	18.33	30	43.33	66.66
MB (this work)	9.1	10.05	11.94	19.51	34.65

Finally, we compare our solution, in terms of communication cost, with the cost required for authenticating equi-joins with the most efficient algorithm presented in [41], i.e., algorithm AIM (see Table 3). In Figure 17 of [41] two relations R and S are equi-joined and the size of the verification object (VO) is displayed, for multiple tuple sizes (a tuple is a row in the relations) $\mathtt{tup} = 32, 64, 128, 256, 512$ bytes. For this experiment, the size of the answer is 31×10^3 tuples and therefore if we use our scheme the cost is $1792 + 20\delta + \delta\mathtt{tup} + 2\delta \times 128$ bytes, for $\delta = 31 \times 10^3$ (the cost $2\delta \times 128$ is due to dealing with duplicate values—see description of equi-join verification before Theorem 3). Note that, especially for large tuple sizes, there are considerable savings with our scheme.

12.3 Verification cost

Let exp, mult, add be the times needed to perform an exponentiation, a multiplication and an addition respectively, all modulo p. Let also EXP, MULT be times required for exponentiation and mulatiplication in group $\mathbb{G}$ and let $\mathcal{EXP}$, $\mathcal{MULT}$ be the respective times in the target group of the bilinear map $\mathcal{G}$. Finally let MAP be the time needed to perform the operation $e(.,.)$. We benchmarked all these operations using the PBC library [1] (version pbc $-$ 0.5.7), on a 64-bit, 2.8GHz Intel based, dual-core, dual-processor machine with 4GB main memory, running Debian Linux, and derived the following times, i.e., MAP $= 5$ms, $\mathcal{MULT} = 0.005$ms, exp $= 0.02$ms, add $= 0.002$ms and mult $= 0.002$ms.

We analyze now the verification cost of a 2-intersection, required by our scheme. Let S_i and S_j be the sets of the intersection. The verification algorithm, on input the proof has to perform the following tasks: (a) First it verifies acc(S_i) and acc(S_j), which requires two bilinear-map computations for each value, therefore taking time 4MAP. (b) Then the experiment of Lemma 3 is executed. The time needed for this part is $\delta(2\text{mult} + 2\text{add} + \text{exp})$; (c) Then the algorithm checks the *subset condition* which takes time 4MAP; (d) Finally it checks the *completeness condition* that takes times $2\text{MAP} + \mathcal{MULT}$. Therefore we see that the total cost for verification of a 2-intersection of size δ is

$$10\text{MAP} + \delta(2\text{mult} + 2\text{add} + \text{exp}) + \mathcal{MULT}\,,$$

which is a linear function in δ, namely the function $50 + 0.028\delta$ (in ms).